Least Authority
PRIVACY MATTERS

Nightfall 3

Security Audit Report

Polygon Technology

Final Audit Report: 27 September 2022

Table of Contents

Overview

Background

Polygon Technology has requested that Least Authority perform a security audit of Nightfall 3, a privacy enhancing application for enabling the shielded transfer of standard ERC20, ERC721, and ERC1155 tokens using zk-SNARKs.

Project Dates

- **April 11 - June 1**: Code Review *(Completed)*
- **June 3**: Delivery of Initial Audit Report *(Completed)*
- **August 16 - September 21**: Verification Review *(Completed)*
- **September 27**: Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Nicole Ernst, Security Researcher and Engineer
- Sven M. Hallberg, Security Researcher and Engineer
- Jasper Hepp, Security Researcher and Engineer
- Anna Kaplan, Cryptography Researcher and Engineer
- Ann-Christine Kycler, Cryptography Researcher and EngineerAC
- Nishit Majithia, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer
- Rai Yang, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of Nightfall 3 followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in-scope for the review:
- Nightfall 3:
  https://github.com/EYBlockchain/nightfall_3

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

```
397a3d02ab28e7b09ac0324c97dab5098d447e4d
```

For the verification, we examined the Git revision:

```
9c6a1e4d12b767bdf34d57ab058ef10e7e263cfc
```

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

Nightfall 3:
https://github.com/LeastAuthority/EYBlockchain_Polygon_Nightfall_3

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:

- `README.md`: https://github.com/LeastAuthority/EYBlockchain_Polygon_Nightfall_3#readme
- Nightfall 3 Documentation: https://github.com/LeastAuthority/EYBlockchain_Polygon_Nightfall_3/tree/main/doc

In addition, this audit report references the following documents:

- M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity." *IACR Cryptology ePrint Archive*, 2016, [AGR+16]
- E. Baker, “Recommendation for Key Management: Part 1 – General.” *NIST Special Publication*, 2020, [Baker20]
- R. Barbulescu and S. Duquesne, “Updating key size estimations for pairings.” *IACR Cryptology ePrint Archive,* 2017, [BD17]
- D. J. Bernstein, M. Hamberg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings." *IACR Cryptology ePrint Archive*, 2013, [BHK+13]
- S. Bowe, A. Gabizon, and I. Miers, "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model." *IACR Cryptology ePrint Archive*, 2017, [BGM17]
- T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." *IEEE Transactions On Information Theory*, 1985, [ElGamal85]
- L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems.” *IACR Cryptology ePrint Archive, Cryptology ePrint Archive*, 2019, [GKR+19]
- J. Groth, “On the Size of Pairing-based Non-interactive Arguments.” *IACR Cryptology ePrint Archive*, 2016, [Groth16]
- T. Kim and R. Barbulescu, “Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case.” *IACR Cryptology ePrint Archive*, 2015, [KB15]
- A. Menezes, P. Sakar, and S. Singh, “Challenges with Assessing the Impact of NFS Advances on the Security of Pairing-based Cryptography.” *IACR Cryptology ePrint Archive*, 2016, [MSS16]
- H. Mirvaziri, K. Jumari, M. Ismail, and Z. M. Hanapi, "Collision Free Hash Function Based on Miyaguchi-Preneel and Enhanced Merkle-Damgard Scheme." *IEEE Xplore,* 2007, [MJI+07]
- T. Perrin, “Curves for pairings.” 2016, [Perrin16]
- Y. Sakemi, T. Kobayashi, T. Saito, and R. Wahby, “Pairing-Friendly Curves.” 2020, [SKS+20]

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the zk-SNARK circuit, smart contract, client, wallet, and Optimist implementations;
- Adherence to the specifications and best practices;
- Common and case-specific implementation errors in the zk-SNARK circuit, smart contract, client, wallet, and Optimist code;
- Adversarial actions and other attacks on the smart contracts, client, and wallet;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;

- Denial of Service (DoS) and security exploits that would impact the code's intended use or disrupt the execution of the code;
- Key management, encryption, and storage;
- Vulnerabilities in the zk-SNARK circuit, smart contract, client, wallet, and Optimist code;
- Protection against malicious attacks and other methods of exploitation;
- Inappropriate permissions and excess authority;
- Performance problems or other potential impacts on performance;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Nightfall 3 provides an ERC20, ERC721, and ERC1155 token transfer API that utilizes Optimistic and zk-SNARKs to counter the high gas costs of direct transactions on the Ethereum blockchain and to allow users to perform transfers that do not reveal any information about the token sent or the recipient (shielded transfers).

Nightfall 3 uses the ZoKrates toolbox to construct Groth16 zk-SNARKs. zk-SNARKs are created to ensure that deposit or withdrawal transactions are valid and to enable shielded transfers. The Nightfall 3 project is inspired by Zcash, using commitments and commitment nullifiers for facilitating shielded transfers with knowledge of the commitment openings being proven in zero-knowledge. In order for a note receiver to be able to spend the note, the commitment opening is encrypted by the commitment spender using the ElGamal encryption scheme over the Baby-JubJub Edwards curve. Correct encryption with the receiver's public key is being proven in zero-knowledge.

The smart contract component manages the on-chain state, with L2 block hashes stored in an array. However, there is no nullifier Merkle tree. Nullifiers are kept in a flat list. `MiMC` hash is used for the commitment Merkle tree. To save gas, not all tree nodes are stored on-chain, but only those required to calculate the new root when a leaf is added.

The Optimist component implements the functionality of the Proposer and Challenger. The proposer creates L2 blocks out of Nightfall user transactions and submits them to the smart contract for state update. The Challenger checks proposed blocks for validity and slashes the Proposer in case of an invalid block proposal.

The client and wallet component implementations both share the same core functionality and serve the purpose of providing the user with an interface to interact with the Nightfall 3 system. The Wallet is browser-based and allows users to create accounts and make transactions. The Client is the library that contains the functionality to interact with Nightfall 3, which is used by the wallet and can be used by developers for integrating Nightfall 3 into their projects.

Our team performed a comprehensive review of the components that comprise the Nightfall 3 system. We found security has been taken into consideration in the design of Nightfall 3, as demonstrated by the implementation of a zero-knowledge proof system that can protect the privacy of users. However, our team identified security issues and many areas of improvement, as detailed below.

System Design

Smart Contract Implementation Issues

We performed a comprehensive review of the Nightfall 3 smart contracts. We found a check is missing in the `Shield` smart contract to prevent an attacker from repeatedly withdrawing a deposited proposer block stake, draining all deposits controlled by the smart contract. We recommend that `isBlockStakeWithdrawn` be set to `true` after a block stake withdrawal (Issue A).

When challenging transactions with duplicate nullifiers, if both transactions have a zero as one of the nullifiers, then the challenge would pass, causing the proposer to be slashed for a valid proposal. We recommend implementing a check that tests if one of the nullifiers is non-zero (Issue B). In addition, the function `challengeHistoricRoot` is used to verify that the proposed block number is correct. There is a check missing to determine if the historic root block number is non-zero in the case of a deposit transaction, or if the second historic root block number is non-zero in the case of single transfer and withdrawal transactions. As a result, an invalid challenge would pass, resulting in the challenger being rewarded and the proposer getting slashed. We recommend implementing a check to verify that the historic root block number is non-zero (Issue C).

Nightfall 3 implements a feature to allow users to withdraw their deposits without having to wait for the block challenge period to end. To do this, the user must pay a liquidity provider a fee to receive an advance on the withdrawal, with the liquidity provider collecting the advanced amount at the end of the block challenge period. This feature is vulnerable to front running whereby `advanceWithdrawal` is front run, resulting in the original liquidity provider missing out on making the advance and earning the fee after having made the necessary validation computations. We recommend a mitigation that requires the withdrawing user to perform the withdrawal after obtaining the signature of the liquidity provider (Issue I).

The functions `getWithdrawInputs` and `getDoubleTransferInputs` expect an incorrect number of inputs, which could cause unintended behavior. We recommend that the functions be corrected to the actual length (Issue J). Moreover, the `Shield` smart contract does not adhere to best practice in the implementation of safe transfer. We recommend using standard libraries (Issue K).

Centralization

In the smart contracts, the security-critical `Config` and `Key_Registry` smart contracts are owned by an Externally Owned Account (EOA). A single private key is more likely to be lost, stolen, or misused, which would disrupt the functionality of the system. At the very minimum, we recommend that a Multi-Sig account be used until a DAO can be put in place (Issue D). The `Config` smart contract owner currently sets a single challenger within the system. An idle or malicious challenger could disrupt critical functionality, leading to the loss of assets. We recommend that the role of the challenger be made decentralized (Issue E). Similarly, the current implementation of a single proposer role requires that users trust a central entity that could censor their transactions or be idle. We suggest a decentralized proposer, which would protect users from this issue (Issue F).

A decentralized challenger role, however, would necessitate a scheme to prevent front running of the challenge. In the case of a commit-reveal scheme, the commit step of the scheme could still potentially be front run by an attacker, preventing a challenger from commiting a challenge within the challenge period. We recommend continued research on ways to mitigate the risk that front running can pose to the functionality of the system (Issue H).

Optimist Incentive Mechanism

In the examination of the incentive mechanism motivating the proposer and the challenger, we found that while the proposer is incentivized to propose valid blocks with transaction fees, and disincentivized (by being slashed) from proposing invalid blocks, the challenger is only rewarded upon correctly challenging invalid blocks. This incentive mechanism requires that the user trust that the challenger role is performed despite the possibility of incurred losses (Issue L). The challenger is also disincentivized from challenging large blocks because of high gas costs. There is no restriction on the number of transactions that a proposer can submit in a single block. A block could be large enough, resulting in the gas cost of a challenge being too high, thereby preventing invalid blocks from being challenged. We recommend that a restriction on the size of the L2 blocks be implemented (Issue G).

zk-SNARK Circuits

Nightfall 3 uses the ZoKrates toolbox to construct Groth16 zk-SNARKs to ensure that deposit or withdrawal transactions are valid and to enable shielded transfers.

The Groth16-based zero-knowledge proof system used by Nightfall 3 requires the computation of a Common Reference String (CRS) in a preprocessing phase. As the system is only secure under the assumption that the simulation trapdoor generated during this computation is deleted, a best practice approach for the execution of this trusted setup phase is to perform the task in a Multi-Party Computation (MPC), as defined in [BGM17]. However, in the current version of the system, the CRS is computed by a single party, which necessitates that this party be fully trustworthy. We recommend performing a proper multi-party trusted setup phase instead (Issue P).

The Nightfall 3 zk-SNARK scheme is implemented over Ethereum's BN254 curve, which is considered to be insufficiently secure, with a security level below the recommended 128-bits. We recommend the use of BLS12-381(Issue Q).

Cryptographic primitives such as ElGamal encryption, the `MiMC` symmetric cipher, and associated hash functions have been implemented by the Polygon Technology team. We compared the Nightfall 3 implementation of the ElGamal encryption scheme to best practice standards and found that the Nightfall 3 implementation exposes the system to small subgroup attacks. Additionally, we looked at the use of the `Elligator_2` construction from [BHK+13] to map messages onto the Baby-JubJub Edwards curve. We found the implementation to be insufficient, as the images might have small cofactors (Issue M). We compared the implementation of the `MiMC` symmetric cypher to its definition [AGR+16] and could not identify any issues. We compared the various `MiMC` hash functions to the requirements from [AGR+16] and the Miyaguchi-Preneel construction [MJI+07] and could not identify any issues. However, we found that the documentation of the derivation of the constants used in the various `MiMC` constructions could be improved (Suggestion 3).

We found that Nightfall 3 inappropriately uses SHA256 as a pseudorandom function (PRF) to compute the nullifier, which could undermine the security assumptions of the system. We recommend that a purpose-appropriate PRF be used, or that the current implementation be modified for this purpose (Issue O). In addition, the Nightfall 3 zero-knowledge proof system utilizes a key schedule whose design can be improved. Although the current implementation has been adapted from Zcash, variable names have not been updated to reflect their modified purpose. In addition, the implemented modifications are difficult to verify cryptographically. We recommend modifications to the implementation of the single and double transfer to remediate this issue (Issue N). Furthermore, the naming convention for the zk-SNARK proof system keys should be updated to reflect the purpose of each key within the Nightfall 3 system (Suggestion 6), and we suggest that a redundant check on the `ask` key be removed when computing a double transfer commitment (Suggestion 1).

Moreover, for deposit and withdrawal transactions, the `value` field in the commitment for those transactions is limited to 64-bits, which, under certain circumstances, could be exceeded. In this case, any withdrawal transaction would fail. We recommend that this restriction be better configured (Suggestion 12).

Nightfall Client Wallet

Our team investigated the wallet and client implementations and found the design to be insufficiently secure. We found that a hash function is used as a general purpose cryptographic function. Within the implementation, hash functions are used to construct a message authentication code (MAC), a key derivation function (KDF), and a commitment scheme. This is not a recommended practice in the design of a secure system because each of these applications have different security requirements.
We identified an issue in the wallet/client ElGamal implementation that occurs when decrypting on-chain ciphertexts, as the decryption algorithm fails to check all possibilities that result from the conversion of a `BN254` scalar field `Fp` element into the intended `u32[8]` commit opening. This could result in the wallet/client not being able to deduce the correct commit opening, preventing the user from spending the note. We recommend that the algorithm be modified to check all possibilities until the proper data is found, or that commit openings be modified in order to prevent ambiguity in the first place (Issue Z).

The `BigInt` type is used to perform cryptographic operations, which can make secret data vulnerable to timing side-channel attacks (SCA). We recommend that cryptographic libraries, such as libsnark, be used to perform cryptographic operations (Issue X). In addition, the JavaScript remainder operator is used for modulus computations performed by the wallet/client on the `BigInt` type, which is unsafe because the JS remainder operator does not compute the correct modulus for negative numbers that are possible representations of prime field elements. As such, its behavior is almost impossible to audit. We recommend using a safe modulus operator or that unsafe modulus computations be replaced by a proper type `Fp` for arithmetics in prime fields (Issue Y).

Moreover, in the current number representation, some calculations performed for gas and transaction costs could exceed the maximum value before experiencing precision loss effects. We recommend using an appropriate number representation to perform these calculations (Issue V).

Our team closely investigated secret key management within the wallet and client implementations and identified several issues that could cause users to lose control of their assets. The wallet stores user private keys encrypted in the browser's local storage. However, the encryption key is also stored in the user file system. In the event that the user file system is compromised, the attacker could trivially extract the encryption key, decrypt the secret material, and take control of the wallet. We recommend that all wallet data be encrypted using a key derived from the account mnemonic (Issue R).

In addition, user private data such as transaction history, commitment data, and others are persisted to the disk, unencrypted. An attacker could extract this data and use it to deanonymize the user or link the user to specific accounts or transactions. We recommend that all private user data be encrypted when not in use (Issue W).

Furthermore, there is no way for users of the Nightfall 3 wallet/client to rotate encryption keys used to encrypt data stored by the browser. We recommend that this functionality be implemented to allow users to change encryption keys as needed (Issue U). We also found that upon the generation of a new wallet, the wallet/client writes the generated keys to the log. Given certain preconditions, an attacker could extract the keys from the logs, taking control of the users' assets. We recommend preventing secrets from being logged (Issue T).

There is no functionality implemented to allow a user to recover an account from a mnemonic phrase. As a result, the user would not be able to move the wallet to another device or, in case of device failure, be able to recover the account trivially. We recommend the implementation of functionality to allow the wallet user to recover an account and commitments using the mnemonic phrase (Issue S).

Furthermore, an arbitrarily limited number of accounts a user may generate prevents users from freely generating accounts to enhance their security. We recommend that the limit be increased or removed

(Suggestion 8). Once these recommended updates are implemented, it is recommended that a follow-up review of the wallet/client cryptographic libraries be performed by an independent security team.

Code Quality

Smart Contracts

The Nightfall 3 smart contracts codebase is generally well organized. However, the quality of the implementation could be improved by removing multiple instances of commented-out and unused code and resolving the many TODOs that remain in the codebase. This would improve the readability of the code and reduce the risk of implementation errors, which could lead to security vulnerabilities. We recommend that unused and commented-out code be removed and that outstanding TODOs be resolved (Suggestion 4).

zk-SNARK Circuit

The Nightfall 3 zk-SNARK circuit implementation code is well organized. The ZoKrates zk-SNARK toolbox is well suited for writing auditable code. However, the toolbox is considered a proof-of-concept implementation that is not ready for production use.

Client and Wallet

Although the Nightfall 3 wallet and client are two different components, they perform the same core functionality and contain duplicate code. As a result, the structure of the codebase is not clear. This can be confusing for reviewers and maintainers and can increase the risk of errors being missed. We recommend avoiding duplicating code in both the wallet and client components and that shared code be abstracted to a single location (Suggestion 5).

We found that in the wallet and client implementations, numbers are represented in inconsistent formats and types. This can be confusing for reviewers and maintainers and increases the risk of bugs or vulnerabilities going unidentified. We recommend that appropriate number representations be selected for the purpose at hand and that these representations be used consistently (Suggestion 9). In addition, we recommend adhering to the best practice of separating user-configurable variables from immutable constants to improve readability and reduce the risk of implementation errors (Suggestion 10).

In both implementations, we found that error handling does not adhere to best practice, as many errors are either ignored or handled improperly. Errors must be propagated appropriately, with the error returning useful information to help the user resolve the error. We recommend improving error handling (Suggestion 7).

Tests

Although Nightfall 3 implements an end-to-end test suite, it does not implement sufficient unit tests. The smart contracts include basic tests but some tests failed to run (e.g., the challenger test `Neg-http.mjs`). Other security-critical functionalities have insufficient test coverage, including the zk-SNARK circuit and the wallet/client implementations. We recommend improving test coverage for Nightfall 3 (Suggestion 2).

Documentation & Code Comments

Our team found the zk-SNARK circuit implementation documentation to be generally accurate and helpful. However, some aspects of the documentation are outdated. To improve the documentation, we recommend updating the names of variables to match their intended functionality in the code and documentation (Suggestion 3). In addition, in the documentation, we advise referencing research publications for definitions of abstract concepts used in the system to improve auditability. The zk-SNARK circuit implementation is sufficiently commented.

The documentation of the smart contracts and wallet/client components of the system was insufficient in comprehensively describing the functionality of each of the components. We recommend that documentation and code comments be improved (Suggestion 11).

Scope

We found the scope of the audit for Nightfall 3 to be sufficient in that it encompassed all security-critical components.

Dependencies

The Nightfall 3 zk-SNARK circuit implementation utilizes the ZoKrates zk-SNARK toolbox. However, the ZoKrates README explicitly describes it as a proof-of-concept not tested for production use. An associated GitHub issue has been inactive since its creation in July 2020.

The Nightfall 3 wallet/client number conversions rely on potentially two levels of unaudited external libraries, `general-number` and `zkp-utils`. We recommend the utilization of well-audited and maintained libraries. We advise that a follow-up security review of the wallet/client component be performed once the recommendations from this report have been implemented.

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
Issue A: Block Stake Can Be Withdrawn Repeatedly [Smart Contracts]	Resolved
Issue B: Invalid Duplicate Nullifier Challenge Can Pass [Smart Contracts]	Resolved
Issue C: Invalid Historic Root Challenge Would Pass [Smart Contracts]	Resolved
Issue D: EOA Config.sol and Key_Registry.sol Owner Account Could Be Compromised [Smart Contracts]	Resolved
Issue E: Centralization of a Single Challenger [Smart Contracts]	Unresolved
Issue F: Liveness and Censorship Issue with Single Proposer [Smart Contracts]	Unresolved
Issue G:L2 Block Could Be Too Costly to Challenge [Smart Contracts]	Resolved
Issue H: Front Running Commit Transaction for Challenge (Out of Scope) [Smart Contracts]	Unresolved
Issue I: Front Running Advance Withdrawal [Smart Contracts]	Unresolved
Issue J: Incorrect Public Input Length for Withdraw and Double Transfer	Resolved

[Smart Contracts]	
Issue K: Unsafe ERC20 Token Transfer [Smart Contracts]	Resolved
Issue L: Verifier’s Dilemma for the Challenger [System Design]	Partially Resolved
Issue M: ElGamal Encryption Is Vulnerable to Small Subgroup Attacks [Circuit]	Resolved
Issue N: Key Schedule in Zero-Knowledge Proof Is Not Well Designed [Circuit]	Resolved
Issue O: Merkle-Damgård Hashes Are Used as Pseudorandom Functions [Circuit]	Resolved
Issue P: Proper Trusted Setup [Circuit]	Unresolved
Issue Q: The BN254 Curve Provides Insufficient Security [Circuit]	Unresolved
Issue R: Local Encryption Key Stored in IndexedDB [Wallet/Client]	Unresolved
Issue S: Wallet Recovery from Mnemonic Is Not Implemented [Wallet/Client]	Partially Resolved
Issue T: Key Material May Be Logged To System Logs [Client]	Resolved
Issue U: Non-Existing Key Rotation Protocol [Wallet/Client]	Unresolved
Issue V: Gas and Transaction Calculation Loses Precision When Exceeding JS’s MAX_SAFE_INTEGER Bounds [Wallet/Client]	Resolved
Issue W: Privacy-Relevant Data Stored Unencrypted on File System [Wallet]	Unresolved
Issue X: BigInt Operations Are Not Constant-Time [Wallet/Client]	Resolved
Issue Y: Unsafe Usage of JavaScript’s Remainder Operator in Modulus Computations [Wallet/Client]	Resolved
Issue Z: Incomplete Ciphertext Decryption Algorithm [Wallet/Client]	Resolved
Suggestion 1: Remove Check on Ask in Double Transfer [Circuit]	Resolved
Suggestion 2: Expand Test Coverage [All Components]	Partially Resolved
Suggestion 3: Document the Source of the MiMC Constants [Circuit]	Resolved
Suggestion 4: Remove Unused and Commented-Out Code [Smart Contracts]	Unresolved
Suggestion 5: Avoid Code Duplication [Wallet/Client]	Unresolved
Suggestion 6: Update Naming of Keys [Wallet/Client]	Resolved
Suggestion 7: Improve Error Handling in Wallet [Wallet/Client]	Unresolved

Suggestion 8: Allow Users to Create More Nightfall Accounts [Wallet/Client]	Unresolved
Suggestion 9: Unify Number Representations [Wallet/Client]	Unresolved
Suggestion 10: Separate Configuration from Constants [Wallet/Client]	Unresolved
Suggestion 11: Improve Nightfall 3 Documentation and Code Comments [All Components]	Unresolved
Suggestion 12: Rewrite Withdrawal Function Making Notes Larger than 64-bits Withdrawable [All Components]	Unresolved

Issue A: Block Stake Can Be Withdrawn Repeatedly [Smart Contracts]

Location
`/nightfall-deployer/contracts/Shield.sol#L65`

Synopsis
The block stake can be withdrawn repeatedly by a malicious proposer as a result of a variable, which is supposed to indicate that a block's stake has been withdrawn, rather than set, after a withdrawal transaction.

Impact
The attacker can repeatedly withdraw their own stake and thereby drain the entirety of the deposited block stakes.

Preconditions
The attacker must be a proposer who has deposited a stake for a block before.

Feasibility
When preconditions are met, executing the attack is trivial since all the attacker has to do is call the `requestBlockPayment` function repeatedly.

Technical Details
The variable `isBlockStakeWithdrawn` in the `state` struct inside the `Shield` smart contract tracks whether a block's stake has been withdrawn. A check is performed on line 65 that reverts in case the stake has already been withdrawn. Because this variable is never set to `true`, this check will always pass, allowing an attacker to call `requestBlockPayment` repeatedly.

Remediation
After the pending withdrawal is added to the state on line 76, we recommend adding the function `setBlockStakeWithdrawn` to correctly set `isBlockStakeWithdrawn` to `true.`

Status
The Polygon Technology team has implemented an assertion that `BlockStakeWithdrawn` has not already occurred.

Verification
Resolved.

Issue B: Invalid Duplicate Nullifier Challenge Can Pass [Smart Contracts]

Location
/nightfall-deployer/contracts/ChallengesUtil.sol#L213

Synopsis
The function `libChallengeNullifier` performs a duplicate nullifier challenge to prevent double-spending by challenging two duplicate nullifiers from two different transactions. If either of the two transactions has a zero nullifier, which is valid for some types of transactions, the challenge would still pass.

Impact
The proposer is slashed and the challenger is rewarded for an invalid challenge.

Preconditions
Either one of the nullifiers in both transactions is zero, which is possible for deposit, single transfer, and withdrawal transactions.

Technical Details
In `libChallengeNullifier`:

The duplicate nullifier check:

```
require(tx1.nullifiers[nullifierIndex1] ==
tx2.nullifiers[nullifierIndex2],'Not matching nullifiers')
```

would pass when:
`tx1.nullifiers[nullifierIndex1]==tx2.nullifiers[nullifierIndex2]`

and both of the nullifiers are zero.

Remediation
We recommend implementing a check to verify that either one of the nullifiers is non-zero:

```
require(tx1.nullifiers[nullifierIndex1]!=0 && tx1.nullifiers[nullifierIndex1]
== tx2.nullifiers[nullifierIndex2],'Not matching nullifiers')
```

Status
The Polygon Technology team has implemented a check to verify that duplicate nullifiers do not exist.

Verification
Resolved.

Issue C: Invalid Historic Root Challenge Would Pass [Smart Contracts]

Location
/nightfall-deployer/contracts/Challenges.sol#L317

Synopsis

The `challengeHistoricRoot` function checks if the historic root of an L2 block number in a transaction is greater than the number of L2 blocks on-chain. The current implementation is missing a check to confirm that the `historicRootBlockNumberL2` is non-zero. An invalid historic root challenge could pass for deposit, single transfer, and withdrawal transactions.

Impact

The proposer is slashed and the challenger is rewarded for an invalid challenge.

Technical Details

For a deposit transaction, the valid challenge condition should be that the `historicRootBlockNumberL2` is non-zero. As a result, the invalid challenge condition that `historicRootBlockNumberL2` is zero would pass. For single transfer and withdrawal transactions, the valid challenge condition should be that the second `historicRootBlockNumberL2` is non-zero.

Remediation

We recommend the check for a deposit transaction be implemented as follows:

```
require( uint256(historicRootBlockNumberL2[0]) != 0 ||
uint256(historicRootBlockNumberL2[1]) != 0, 'Historic root does not exist').
```

For the final `else` check for single, transfer, and withdrawal transactions, we recommend the following steps:

```
require(state.getNumberOfL2Blocks() < historicRootBlockNumberL2[0] ||

historicRootBlockNumberL2[1] != 0, 'Historic root exists')
```

Status

The Polygon Technology team has modified the historic root challenges as recommended.

Verification

Resolved.

Issue D: EOA Config.sol and Key_Registry.sol Owner Account Could Be Compromised [Smart Contracts]

Location

`/nightfall-deployer/contracts/Config.sol`

`/nightfall-deployer/contracts/Key_Registry.sol`

Synopsis

The owner of the `Config` and `Key_Registry` smart contract is an EOA (currently the deployer), which means that a single private key controls the smart contracts. A single private key is more likely to be lost, stolen, or used maliciously. Should this EOA be compromised, the attacker can cause harm to the network.

Impact

The attacker could make themself the only proposer and challenger or register a malicious verifier key to verify invalid proofs, thereby disrupting the intended behavior of the system.

Preconditions

The attacker must compromise the owner's private key.

Feasibility

The compromise of a single private key is possible.

Technical Details

By compromising the owner's private key, the attacker gains control of the `Config` smart contract. `setBootChallenger` can be called to set the boot challenger and `setBootProposer` can be called to set the boot proposer, controlling the single challenger and single proposer in the system. The attacker could censor transactions or freeze user funds. Also, `registerVerificationKey` could be called to register a malicious verification key to verify invalid transaction proofs and steal user funds.

Remediation

We recommend using a Multi-Sig account for the owner of the `Config` and `Key_Registry` smart contract. We also recommend that the ownership of the smart contract be ultimately delegated to a DAO.

Status

The Polygon Technology team has implemented a Multi-Sig account in accordance with the recommendation.

Verification

Resolved.

Issue E: Centralization of a Single Challenger [Smart Contracts]

Location

`/nightfall-deployer/contracts/Challenges.sol`

`/nightfall-deployer/contracts/Config.sol#L42`

Synopsis

In the current implementation, a single address is set to perform the role of the challenger. This `bootChallenger` is configurable by the owner of the `Config` smart contract (see Issue D). This implies that the entire system is reliant on the availability and honesty of a single and owner-appointed challenger.

Impact

An idle or malicious challenger could prevent an invalid L2 block containing invalid transactions from being challenged before being finalized, which can result in the loss of funds.

Preconditions

The boot challenger is offline or malicious.

Remediation

We recommend that the challenger role be decentralized by allowing anyone to challenge an invalid L2 block. As long as there is one honest challenger, the invalid block, with invalid transactions included in the block, will be challenged.

Status

At the time of the verification, the implementation of the recommendations remains pending.

Verification

Unresolved.

Issue F: Liveness and Censorship Issue with Single Proposer [Smart Contracts]

Location

`/nightfall-deployer/contracts/Proposers.sol#L38`

`/nightfall-deployer/contracts/State.sol#L69`

Synopsis

Currently, only the proposer (boot proposer deployed by Nightfall 3) exists. The user must trust the boot proposer to act honestly when submitting transactions. In case the proposer is idle, offline, or censoring the transaction, the L2 block will not be created or transactions submitted by the user will not be included in an L2 block, locking user funds in the `Shield` smart contract.

Impact

User transactions are censored, and user funds are locked in the `Shield` smart contract. Both deposit and withdraw transactions can be affected.

Preconditions

The single proposer is idle or censoring a transaction.

Technical Details

When a single proposer goes idle or acts maliciously to censor a user's transaction, the transaction submitted (either on-chain or off-chain) will not be included in L2 blocks, and thus will not be processed. In case of a deposit, a user submits the transaction on-chain to the `Shield` smart contract and transfers the funds to the contract, but the deposit transaction submitted on-chain does not get included by the malicious or idle proposer in an L2 block. Since the on-chain state is not updated and published (commitment root, etc.), there is no way to withdraw the deposited funds, as there is no way to get the updated on-chain state. As a result, the funds are locked in the contract, and the same applies for withdraw and transfer transactions as well.

Remediation

We recommend that the Polygon Technology team decentralize the proposer so that anyone can register to be a proposer. In this case, if a proposer goes idle or censors the transaction, as long as there is one honest proposer, the transaction will be submitted on-chain and included in an L2 block. For off-chain transactions, the user could broadcast the transaction to all proposers, significantly reducing the risk of censorship caused by a single proposer.

Status
At the time of the verification, the implementation of the recommendations remains pending.

Verification
Unresolved.

Issue G: L2 Block Could Be Too Costly to Challenge [Smart Contracts]

Location
/nightfall-deployer/contracts/State.sol#L69

/nightfall-deployer/contracts/Challenges.sol

Synopsis
When proposing a block, there is no limit to the number of transactions in the block that a proposer can submit. A large block size would increase the gas cost of a challenge, reducing the incentive for challengers to challenge invalid blocks.

Impact
In the event that an L2 block is too costly to challenge, an invalid block might be finalized without being challenged, which could lead to the loss of funds.

Preconditions
The block is too big to challenge.

Technical Details
A challenge is submitted with the challenged block transaction data as `calldata`, which costs gas. The bigger the block is, the higher the gas cost for the challenger will be.

Remediation
We recommend setting a limit for the L2 block size when the proposer is submitting the block or setting a gas limit for the challenge transaction. If the limit is exceeded, the proposer submitting the block is penalized.

Status
The Polygon Technology team has set a maximum to limit the size of L2 blocks.

Verification
Resolved.

Issue H: Front Running Commit Transaction for Challenge (Out of Scope) [Smart Contracts]

Location
/nightfall-deployer/contracts/Challenges.sol#L370

/nightfall-deployer/contracts/Challenges.sol#L219

Synopsis

A challenge to a proposed block is protected from front running by a commit-reveal scheme. The challenger must make a commit transaction (`commitToChallenge`), then reveal the challenge data pre-image in a subsequent challenge transaction. In the current implementation, the `commit` and challenge can only be called by the `bootChallenger` and, therefore, cannot front run.

If anyone is able to be the challenger, a front runner may front run the `commitToChallenge` transaction repeatedly over the challenge period, preventing other challengers from performing a commit transaction and subsequent challenge transactions.

Impact

The invalid block will not be challenged, as legitimate challengers are blocked from challenging. Blocks containing invalid transactions going unchallenged would lead to the loss of user funds.

Preconditions

The front runner is able to front run every commit transaction over the 7-day challenge period.

Feasibility

The attack is difficult but feasible in the event that the front runner wants to disrupt the system or collude with a malicious proposer to gain profit from proposing invalid transactions.

Technical Details

When a challenger commits `(CommittedToChallenge)` to a challenge, they will wait to receive a `CommittedToChallenge` (`commitHash`, `msg.sender`) event (where `msg.sender` is their address) before submitting a challenge (with the challenge data pre-image) transaction. If their commit is front run, then the emitted event will not contain their `msg.sender` and, so, they will not be able to submit their reveal. A challenger can send a new commit and if all the commits front run repeatedly over the 7-day challenge period, then the challenge gets blocked and the invalid block is finalized.

Remediation

We suggest that the Polygon Technology team keep researching alternative ways to mitigate front running attacks.

Status

The Polygon Technology team has acknowledged the concern and stated that long-range attacks are an Ethereum protocol layer issue.

Verification

Unresolved.

Issue I: Front Running Advance Withdrawal [Smart Contracts]

Location

`/nightfall-deployer/contracts/Shield.sol#L169`

Synopsis

A front runner can front run the advance withdrawal (`advanceWithdrawal`) transaction whenever it is called and take the liquidity provider opportunity to make an advance and earn the withdrawal fee after having performed the validation computation. The front runner has to pay the fund to the user first and wait for the challenge period to pass to withdraw their fund paid to the user in advance.

Impact

Front running of `advanceWithdrawal` could remove the incentive for liquidity providers to make the instant withdrawal available for users.

Preconditions

A liquidity provider calls `advanceWithdrawal` to immediately fund a recipient user, with the liquidity provider waiting for the challenge period to pass on their behalf.

Feasibility

The feasibility of this attack depends on the total withdrawal fees in comparison to the gas costs of the attack.

Technical Details

In `advanceWithdrawal`:

After the liquidity provider transfers the token to the user, the withdrawal fee is transferred to the liquid provider (`msg.sender`), which can be front run:

```
state.addPendingWithdrawal(msg.sender, advancedFee);
```

Mitigation

There is no known effective protection against front running in general. As a result, we recommend mitigating this issue by removing the use of `msg.sender` and having the recipient user explicitly use the advance withdrawal message signed by the liquidity provider to receive payment.

Status

The Polygon Technology team has stated that they intend to implement the recommended mitigation in the future. Hence, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue J: Incorrect Public Input Length for Withdraw and Double Transfer [Smart Contracts]

Location

`/nightfall-deployer/contracts/Utils.sol#L137`

`/nightfall-deployer/contracts/Utils.sol#L148`

Synopsis

The input length for public input for proof verification in the function `getWithdrawInputs` is incorrectly set to 16 instead of 6, and in the function `getDoubleTransferInputs`, is incorrectly set to 6 instead of 16.

Impact

This could result in unintended behavior of the functions.

Technical Details

In `getWithdrawInputs`, the inputs array length is set to 16 instead of 6:

```
uint256[] memory inputs = new uint256[](16);
```

In `getDoubleTransferInputs`, the input length is set to 6 instead of 16:

```
uint256[] memory inputs = new uint256[](6);
```

Remediation

We recommend that the input length be set correctly.

Status

The Polygon Technology team has constrained the array lengths appropriately in accordance with the recommended remediation.

Verification

Resolved.

Issue K: Unsafe ERC20 Token Transfer [Smart Contracts]

Location

`/nightfall-deployer/contracts/Shield.sol#L202`

`/nightfall-deployer/contracts/Shield.sol#L256`

`/nightfall-deployer/contracts/Shield.sol#L289`

Synopsis

ERC20 tokens `transfer` and `transferFrom` are unsafe methods for token transfers that do not adhere to the IERC20 standard of reverting in the case of failure. In some cases, these methods may cause tokens to be locked in the contract.

Impact

User funds may potentially get locked in the contract.

Remediation

We recommend the use of the standard safe transfer wrapper by OpenZeppelin.

Status

The Polygon Technology team has started using the safe transfer library to perform transfers.

Verification

Resolved.

Issue L: Verifier's Dilemma for the Challenger [System Design]

Location

`/main/nightfall-deployer/economics.md`

Synopsis

The incentive structure is such that the challenger is only rewarded for correctly challenging an invalid proposed block. Therefore, the verifier's dilemma occurs when an active challenger exists, and no one proposes an incorrect block to avoid getting slashed, eventually resulting in the challenger no longer having an incentive to do its job. As a result, it is reasonable to assume that the challenger stops investing in challenging blocks due to the behavior of the proposers and the computational costs associated with challenging. An idle challenger can lead to a state where users can double-spend transactions, and proposers can introduce invalid transactions. Potentially, all funds locked in the `Shield` smart contract can be stolen if the challenger role is not performed diligently.

Impact

A malicious proposer that is not challenged could drain all the assets in the `Shield` smart contract. On one hand, this issue might seem unlikely since the Polygon Technology team has an incentive to ensure the security of its system. On the other hand, the impact of this attack is severe, and, as a result, users are forced to trust the challenger.

Mitigation

The idea described here lays out a possible approach to solve this problem. In short, the proposer sends an attention challenge from time to time to the challenger. The proposer computes f(x) and posts x together with an encrypted challenge. The challenger needs to respond correctly by computing f(x) and posting it on-chain if $f(x) < T$ in a given time window. If the challenger does not respond, it gets punished. We recommend implementing this approach, as it ensures that the challenger is active as long as the cost of computation is lower than the cost of getting punished.

Status

The Polygon Technology team perspective is that the current incentive structure is sufficient to ensure a well-functioning system and, specifically, the existence of a well-incentivized challenger. While we agree that this argument is intuitively convincing, the possibility of an idle challenger remains. We recommend the Polygon Technology team perform a rigorous analysis of the incentives of challengers and continue to monitor the ongoing debate.

Verification

Partially Resolved.

Issue M: ElGamal Encryption Is Vulnerable to Small Subgroup Attacks [Circuit]

Location

`/common/encryption/el-gamal4.zok`

Synopsis

The security of the ElGamal encryption scheme is based on the assumption that the underlying cyclic group has no subgroups of small order. However, the ElGamal system implemented in Nightfall 3 is defined over the group of rational points of the Baby-JubJub curve, which is not of prime order and therefore has small subgroups.

Impact

The system is vulnerable to small subgroup attacks that might recover information about the messages.

Preconditions

No preconditions are necessary.

Feasibility

Attacks like these are well known and easy to execute.

Technical Details

For a summary of small-order attacks on elliptic curves of non-prime order, see reference.

Mitigation

To mitigate the leakage of partial message information, a possible solution would be to modify the Elligator map implementation in the wallet in such a way that it maps to the large prime order subgroup only, and then enforce the large order subgroup membership in the circuit for an Elligator image `M` and prime order p by

```
assert(p*M == 0)
```

This idea can be instantiated, under the assumption that every plaintext is guaranteed to either consist of less than 256-bits or be a random value, with the unused bits that could be interpreted as a counter to implement a try-and-increment loop, where the counter is increased every time Elligator does not map onto the large prime order subgroup. On decryption, those bits are then stripped from the message.

Note that not all plaintexts have space for additional randomness. To resolve this, we recommend either splitting the respective plaintext into two field elements (increasing the costs for required storage since both need to be encrypted) or rearranging the bits of all plaintexts so all plaintexts can have space for randomization. Since the `ercAddr` is only 160-bits long, all other plaintexts can move the top 16-bits to the plaintext containing the `ercAddr`, resulting in all plaintexts having enough space for randomness, e.g., counters.

Remediation

In section 2.3 of [BHK+13], the authors introduce an asymmetric encryption scheme based on the KEM-DEM structure that works well with Elligator functions. KEM-DEM constructions perform hybrid encryption by first encrypting a random key using asymmetric means and then using symmetric encryption to encrypt the data.

The scheme described in the paper requires a hash function and a symmetric encryption function. For the hash function, we recommend using `MiMC` [AGR+16] or `Poseidon` [GKR+19]. Both are provided in the ZoKrates standard library. Since the ZoKrates standard library does not provide encryption algorithms, we suggest using a different hash `H` in counter mode. This encryption function can encrypt sequences of field elements using a single key and a nonce, therefore decreasing the amount of on-chain storage space for the ciphertext. To encode the `u32[8]` plaintext as field elements, we recommend that the Polygon Technology team encode the top 8-bits of `token_id`, `token_value` and `salt` together with the 160-bit `ercAddr` in one field element, and then encode the remaining lower 248-bits for `token_id`, `token_value` and `salt` as three additional field elements. This way, no ambiguity in the conversion between `u32[8]` and the field type occurs. The encryption of the i'th plaintext block p_i using key k is:

$$c_i = H(k+i) + p_i.$$

The full ciphertext is $(R,\ c_{0..n})$ where R is the ephemeral public key. Note that the two hashes should not be identical. To construct different hashes from a single hash function, we recommend that the Polygon Technology team prepend a field element that describes the specific use of the hash function in this context. After that, a constant can be defined for the use of encryption and prepended to the data being hashed, e.g.:

```
DOMAIN_KEM = to_field(SHA("nightfall-kem")[0..248])
DOMAIN_DEM = to_field(SHA("nightfall-dem")[0..248])
H_KEM(x) = H(DOMAIN_KEM, x)
H_DEM(x) = H(DOMAIN_DEM, x)
```

where H_{KEM} is used to hash the shared secret (in order to make it uniformly random) and H_{DEM} is used for generating the key-stream in counter-mode encryption.

Status

The Polygon Technology team has implemented the KEM-DEM construction as recommended in the remediation of this issue.

Verification

Resolved.

Issue N: Key Schedule in Zero-Knowledge Proof Is Not Well Designed [Circuit]

Location

`/nightfall-deployer/circuits/single_transfer.zok#L62-L68`

`/nightfall-deployer/circuits/double_transfer.zok#L62-L68`

Synopsis

The key schedule used in the zero-knowledge proof is inspired by that of Zcash but modified to a degree that the intended security of the individual keys is difficult to understand and reason about rigorously. Specifically, the names of variables do not describe their purpose, and the use of `ask` for enforcing the need of a "second secret" is highly non-standard and better solved outside the circuit.

Impact

The soundness of the cryptographic design is hardly verifiable.

Technical Details

The design process of Nightfall 3 appears to be a simplified version of Zcash Sapling modified to meet the minimum feature requirements for Nightfall. Although this is a reasonable strategy, in this case, it has led to suboptimal results.

The variable names of Zcash have been kept, but, to a large degree, they do not serve the purpose they initially had in Zcash. This leads to confusing naming. For example, `ask` used to be an "authorization secret key," but in Nightfall 3, it is just an additional secret value that needs to be passed into the circuit and is only used in the derivation of `pkd`. The variable `pkd`, in turn, used to be pk_d in Sapling, which stands for "a public key, diversified with value d". Nightfall 3 does not support diversification in this way, so the name, once again, does not really describe the variable very well.

Another problem is the use of `ask`. The value has been described by the Polygon Technology team as a method to require an additional secret necessary for sending assets that can be stored separately for added security. However, the value is barely used in the circuit and the method is not standard. A standard solution to require input from multiple computers would be to use a single key in the circuit, split it into multiple shares (using, e.g., Shamir secret sharing), and keep the shares on different devices. To compute a proof, these devices would send shares to the prover, which would then reassemble the key and compute the proof. Using secret sharing has the added benefit that it is more flexible, as it not only supports requiring exactly two keys, but also any number that is configured up front. It even supports threshold recovery, where k out of n keys are required for reassembling.

Remediation

We recommend:

- Removing `ask`
- Adding `rk` (root key)
- Renaming `nsk` to `nk` (nullifying key)
- Renaming `ivk` to `dk` (decryption key)
- Renaming `pkd` to `ek` (encryption key)

These keys have the following computational relationships:

```
rk = BIP44(wallet_root_key, path)
shares = SecretShare.Split(rk, params)
rk = SecretShare.Recover(shares)
dk = PRF(rk, INFO_ENC_KEY)
ek = g^dk
nk = PRF(rk, INFO_NULL_KEY)
nullifier = PRF(nk, commitment_hash)
```

Here, sk is passed into the circuit and the circuit constrains these relationships (except for BIP44 and the secret sharing). The PRF used needs to be domain-separated, ideally using distinct prefixes. We recommend using `Poseidon` (see Issue O).

Status

The Polygon Technology team has implemented the recommended remediation.

Verification

Resolved.

Issue O: Merkle-Damgård Hashes Are Used as Pseudorandom Functions [Circuit]

Location

`/nightfall-deployer/circuits/single_transfer.zok#L97-L103`

`/nightfall-deployer/circuits/single_transfer.zok#L120`

`/nightfall-deployer/circuits/double_transfer.zok#L112-L115`

`/nightfall-deployer/circuits/double_transfer.zok#L126-L132`

`/nightfall-deployer/circuits/double_transfer.zok#L142-L148`

`/nightfall-deployer/circuits/double_transfer.zok#L159-L166`

Synopsis

The Polygon Technology team uses the `SHA256` hash as a pseudorandom function (PRF). This is non-standard and is generally not advised. We propose to either replace the function with a function designed to be a PRF or take additional measures towards making `SHA256` behave like a PRF.

Impact

Nullifiers may be length-extended in order to forge commitments.

Technical Details

The nullifier needs to be computed using a PRF. However, in Nightfall 3, the nullifier is the hash of the concatenation of `nsk` and the hash of the commitment that is being nullified. `SHA256` is a Merkle-Damgård hash. Such hashes cannot usually be used as PRFs since they allow length-extension attacks. If the system somehow guarantees that no two hashes $H(m_1)$ and $H(m_2)$, with m_1 being a prefix of m_2, are ever computed, such attacks cannot happen. In the case of Nightfall 3, however, there is no guarantee that a nullifier is not a prefix of a commitment.

Remediation

Our team can suggest two recommendations. Either one of them would solve the immediate threat but when employed together, they provide solid defense-in-depth.

Firstly, we recommend that the Polygon Technology team use a sponge-based hash function like `Keccak` or `Poseidon`. Hash functions using this design are not susceptible to length-extension attacks and the argument that `H(K || m)` is a PRF is more convincing. There is a tradeoff regarding which function to use. `Keccak` is used in SHA3 and has withstood significant scrutiny but is not very efficient in-circuit. `Poseidon` is more recent and has received less scrutiny but is very efficient in-circuit. It is also the designated hash function for Zcash Orchard, and is therefore tested and relied upon by the ZK community at large. Both functions are included in the ZoKrates standard library.

Secondly, we recommend domain-separating the hashes by adding a prefix identifying whether the message is hashed for nullifying or committing. This prefix can be a single bit. This ensures that a nullifier can never be length-extended to a commitment since no commitment has a preimage starting with the nullifier prefix.

Status

The Polygon Technology team has started using a Poseidon-based PRF for hashing the concatenation of the key and message. This resolves the specific security issue identified in our review. However, domain separation has not been implemented, which we suggested as comprehensive defense. That said, we estimate the probability of this leading to issues to be very low.

Verification

Resolved.

Issue P: Proper Trusted Setup [Circuit]

Location

`/nightfall-deployer/src/circuit-setup.mjs#L64`

Synopsis

Nightfall 3 utilizes a Groth16-based SNARK proof-system [Groth16]. Groth16 produces preprocessing SNARKs and their security is based on the existence of a trusted third-party to compute the CRS in a preprocessing trusted setup phase. In the current design, the executor of the trusted setup is a single entity, which requires that party to be fully trustworthy to delete the simulation trapdoor.

Impact

The simulation trapdoor can be used to generate forged proofs, undermining the security of the entire system.

Preconditions

The trusted setup phase is executed by a single party only.

Feasibility

The attack is straightforward and well known.

Remediation

We recommend investing in the development of a proper MPC to generate the CRS. Proper and well-tested methods like the Powers of Tau MPC [BGM17] are known and can be executed in a permissionless way.

Status

The Polygon Technology team has stated that they intend to implement the recommended remediation in the future. Hence, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue Q: The BN254 Curve Provides Insufficient Security [Circuit]

Location

`/src/services/generateKeys.mjs#L37-L44`

`/src/zokrates-lib/setup.mjs#L22`

Synopsis

Nightfall 3 uses a variant of the BN254 curve. 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 required by NIST for new products.

Impact

Using the BN254 curve undermines the security of the zk-SNARK scheme, such that the feasibility of computing forged proofs that check as valid cannot be ruled out. Such proofs would pass validation, yet violate the constraints imposed by the circuit.

Preconditions

No preconditions are necessary.

Feasibility

The exact feasibility is difficult to estimate. However, the potential gains from a successful attack are high, which suggests that an attacker would have incentive to invest significant resources.

Technical Details

Attacks based on the Tower Number Field Sieve (TNFS) and the derivatives exTNDS and SexTNFS have led to a reduced estimate of the security level of BN254. The several scholars and practitioners working on this issue do not entirely agree on the new estimate, but opinions range from 96-bits to 110-bits [KB15, MSS16, BD17, Perrin16]. 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 [Baker20, Table 4], which BN254 does not achieve.

Remediation

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

However, there is currently no efficient method on Ethereum 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 Polygon Technology team continue to monitor developments with EIP-2537.

Status

The Polygon Technology team has agreed with the issue of BN254 offering insufficient security. However, the team has responded that the situation cannot be improved within the timeframe of the verification phase of this review. We agree with their conclusion and recommend monitoring the development of EIP-2537.

Verification

Unresolved.

Issue R: Local Encryption Key Stored in IndexedDB [Wallet/Client]

Location

`wallet/src/utils/lib/key-storage.ts`

Synopsis

Account data in local storage is encrypted using a randomly generated "browser key," which is stored in local storage. An attacker with access to the user's file system or a malicious browser extension can recover the key material and compromise the user's accounts.

Impact

This would result in account compromise, leading to the loss of funds.

Preconditions

The attacker must have access (user privilege) to the local file system or have the target user install an attacker-controlled malicious browser extension.

Feasibility

An attacker with technical knowledge will be able to obtain key material, given the preconditions are met.

Technical Details

The browser key is saved to IndexedDB in the form of a `CryptoKey` object. While this object is by design opaque to JavaScript running in the browser, storing it saves its internal data (i.e., the key material) to the file system. Chrome stores the IndexedDB in the form of a LevelDB database using a Chrome-specific comparator (`idb_cmp1`). Open source tools for reading and dumping databases, such as leveldb-cli, can be used to extract the browser key.

The implementation of `CryptoKey` and the usage of the non-extractable flag do prevent content-scripts from extracting the key material or from making the object usable in a background script or service worker of a malicious extension. However, injecting JavaScript into the html of the victim site is still possible. Therefore, a malicious browser extension can inject a script module into the html of the browser wallet, which would recover the `CryptoKey` object from the IndexedDB, use it to decrypt the `ZkpAccount` objects, and send it to the malicious extension.

Mitigation

The browser's local storage (IndexedDB) must be regarded as highly sensitive and access must be restricted accordingly. We recommend encrypting the file system, including any backups, with a sufficiently strong passphrase.

Remediation

We recommend deriving the browser key from a user-supplied password using a memory-hard key derivation function like `argon2id`. This eliminates the necessity of storing the `CryptoKey` object in persistent storage.

Status

The Polygon Technology team has stated that they intend to implement the recommended remediation in the future. Hence, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue S: Wallet Recovery from Mnemonic Is Not Implemented [Wallet/Client]

Location

`wallet/src/views/wallet/index.jsx`

Synopsis

A flow for the recovery of a previous wallet from a known mnemonic phrase is not present in the frontend or supporting code. A loss of the browser's local storage makes it impossible for a user to access their accounts. Additionally, there is no functionality to recover commitment data from the blockchain. If the locally stored commitment data is lost, there is no possibility for the user to restore it with the browser wallet.

Impact

Non-technical users would be unable to access their funds if the local wallet data is corrupted or deleted.

Technical Details

There is no functionality implemented to recover an account from a mnemonic. The function `deriveAccounts` is only called upon the generation of a new wallet. Furthermore, functionality to recover user commitments (i.e., ZKP funds) from the blockchain after their loss from local storage is not present.

Mitigation

The browser used, in particular its local storage (IndexedDB), must be regarded as critical data and backed up accordingly.

Remediation

We recommend the implementation of functionality to recover an account from a mnemonic phrase, including the recovery of user commitments from the blockchain.

Status

The Polygon Technology team has implemented functionality, enabling the recovery of accounts and commitments from a user's mnemonic phrase. However, it requires the user to have a self-made backup of the commitment data, rather than be able to recover commitments from the blockchain, as recommended in the remediation of this issue.

Verification

Partially Resolved.

Issue T: Key Material May Be Logged To System Logs [Client]

Location

`/nightfall-client/src/routes/generate-keys.mjs`

Synopsis

On log level "silly," generated keys are written to the log, potentially exposing them to unauthorized access. When not set to "silly" by default, the log level is user-configurable via the environment and is susceptible to accidental misconfiguration or malicious tampering.

Impact

An attacker with access to the system logs may obtain key materials, which would result in the loss of funds.

Preconditions

An attacker has access to the system logs and can influence the environment variables of the Nightfall 3 client process.

Feasibility

If the preconditions are met, the attack is trivial.

Remediation

We recommend deleting the logging statements in question.

Status

The Polygon Technology team has disabled the logging of keys into system logs.

Verification

Resolved.

Issue U: Non-Existing Key Rotation Protocol [Wallet/Client]

Location

`/src/utils/lib/key-storage.ts#L90-L114`

Synopsis

Nightfall 3 users cannot change the `CryptoKey` object used to encrypt data stored in the browser, as the `keyRotation` function is not used, and its implementation is incomplete.

Impact
The user is unable to update the encryption keys. As a result, if the encryption key has been compromised or revealed to an attacker, the secrets will be usable by an attacker as long as the web wallet account is present on the user's system.

Technical Details
The `keyRotation` function is not being used anywhere in the codebase. Furthermore, the `keyRotation` function does not update reference values stored in local storage after updating the `CryptoKey` and re-encrypting values stored in the browser's IndexedDB. Thus, users have no control over the key rotation process.

Remediation
We recommend completing the `keyRotation` implementation and allowing users to invoke the process when needed.

Status
At the time of the verification, the implementation of the recommendations remains pending.

Verification
Unresolved.

Issue V: Gas and Transaction Calculation Loses Precision When Exceeding JS's MAX_SAFE_INTEGER Bounds [Wallet/Client]

Location
Non-exhaustive list:

`/src/components/BridgeComponent/index.jsx#L166`

`/src/components/BridgeComponent/index.jsx#L193`

Synopsis
Some calculations can exceed the `MAX_SAFE_INTEGER` when using number representations, which may lead to undesired results.

Impact
Incorrect calculations could result in unintended behavior.

Technical Details
JavaScript can safely represent integers between -(2^53 - 1) and 2^53 - 1. Mathematical operations on numbers that exceed those bounds will yield wrong results.

For example in:

```
transferValue * 10 ** token.decimals
```

We have observed that some of the tokens used have 18 decimals, which leads to the aforementioned expression evaluating a number that is not safe.

Remediation

We recommend the use of a number representation that can accommodate large values, such as BigInt().

Status

The Polygon Technology team has implemented the recommended remediation.

Verification

Resolved.

Issue W: Privacy-Relevant Data Stored Unencrypted on File System [Wallet]

Location

`wallet/src/nightfall-browser/services/commitment-storage.js`

`wallet/src/nightfall-browser/services/database.js`

Synopsis

Commitment data, transaction history, and other security-sensitive data are stored unencrypted in the IndexedDB of the wallet. Any privacy-relevant data as well as all data that could influence the wallet's state should be encrypted in persistent storage.

Impact

An attacker with access to transaction history or commitment details could link commitments, transactions, and different addresses to the user.

Preconditions

The attacker has access to the IndexedDB either by controlling an installed malicious extension or by gaining access to the file system.

Feasibility

If the preconditions are met, an attacker can trivially obtain the privacy-relevant information.

Remediation

All information that could link commitments, transactions, and addresses to the user as well as any other privacy-relevant data or data that could influence the wallet state should be encrypted in persistent storage. We recommend implementing the suggested remediation from Issue R that utilizes a user-supplied password to derive an encryption key, which can then be used to encrypt all secrets as well as all privacy-relevant data.

Status

The Polygon Technology team has stated that they intend to implement the recommended remediation in the future. Hence, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue X: BigInt Operations Are Not Constant-Time [Wallet/Client]

Location

/crypto/encryption/elgamal.js

/crypto/encryption/elgamal.mjs

Synopsis

Cryptographic operations are implemented using the standard `BigInt` type, which is not specifically designed for such use. In particular, its methods do not guarantee constant run-time. This exposes the application to the risk of timing side-channel attacks.

Impact

A successful side-channel attack may expose key material and thus allow arbitrary access to account funds.

Preconditions

An attacker must be able to observe, even partly or indirectly, the run-time of operations involving secret data.

Technical Details

Timing measurements could be exposed, for example, via a malicious browser extension in the case of the wallet or an otherwise harmless HTTP endpoint in the case of the client. Practical timing side-channel attacks have been demonstrated under a variety of conditions, including across network connections with significant latency.

Remediation

We recommend that all cryptographic operations be performed using a library specifically designed and vetted for such a purpose, including the requirement that operations run in constant time.

Status

The replacement of ElGamal encryption with KEM-DEM, in accordance with the recommended mitigation of Issue M, makes this issue redundant.

Verification

Resolved.

Issue Y: Unsafe Usage of JavaScript's Remainder Operator in Modulus Computations [Wallet/Client]

Location

/crypto/encryption/elligator2.js

/crypto/encryption/elgamal.js

Synopsis

The Nightfall 3 wallet/client code represents prime field elements as integers of type `BigInt` and computes the associated prime field arithmetics using the remainder operator (%) on `BigInt`. However, in order for this arithmetic to be well defined, the action of the remainder operator (%) should be independent of the representation of the prime field elements, which is not the case.

In fact, a prime field element n from F_p is represented by the set `{n+ k*p | for all integers k}` and, in particular, some representatives are negative numbers. However, according to [JavaScript reference documentation](), the JavaScript remainder operator behaves differently from the mathematical modulus operator on negative integers. As a result, using the remainder operator (%) in computations (see [example]()) is not safe, as it gives different results for different `BigInt` representations of prime field elements.

Impact

Using the remainder operator instead of a proper modulus definition makes auditing the code unnecessarily complicated, as edge cases and case distinctions about the `BigInt` representatives of the prime field elements have to be traced through all paths in the relevant code. In the worst case scenario, the implementation provides wrong cryptographic primitives.

Remediation

We recommend implementing a type Fp for arithmetics in prime fields or, at a minimum, implementing a modulus operator that, for `BigInt` a and n, computes the mathematical operation a `mod` n appropriately, in addition to replacing any occurence of modular arithmetic with that operator. `BigInts` must not be assumed to be positive. Specifically:

```
function mod(a, n) {
  assert(n > 1)
  return ((a % n ) + n ) % n
}
```

Status

The Polygon Technology team has implemented a safe modulus operator.

Verification

Resolved.

Issue Z: Incomplete Ciphertext Decryption Algorithm [Wallet/Client]

Location

[`/nightfall-browser/classes/secrets.js#L72`]()

Synopsis

In order to transfer shielded notes, the sender is required to encrypt the associated commit openings and store their ciphertext on-chain. Openings are of type `u32[8]` and a critical step in the encryption process is the conversion from `u32[8]` into elements from the BN254 scalar field Fp. However, since the modulus p of that field is a 254-bit prime number, the conversion function is not injective, as up to 6 elements of type `u32[8]` can be mapped onto the same element from Fp.

To further elaborate, consider the integers 0, p, 2p, 3p, 4p, 5p. All of them are presentable within 256-bits but map to the same element 0 from Fp. On the other hand, in order for a note to be spendable, the recipient must be able to decrypt the correct commit openings from the on-chain ciphertext. An essential step in the decryption algorithm is the conversion of elements from the BN254 scalar field back into elements of the `u32[8]` type. However, as explained above, the conversion is not unique and any element from Fp can have up to six preimages in `u32[8]`.

The Nightfall 3 wallet/client does not consider all possible 6 cases and, as a consequence, might decrypt the ciphertext into the wrong commit openings.

Impact

Commit openings consist of four components called `ercAddress`, `tokenID`, `token_value` and `salt`. Of those components, `tokenID` and `salt` are expected to be representable with not less than 254-bits in general. However, in those cases, the wallet/client will compute the wrong commit openings with a high probability, rendering the note unspendable.

Remediation

Since the `ercAddress` as well as the `token_value` are restricted to less than 254-bits, the wallet/client needs to loop through all 6 possible cases for `tokenID` and all 6 possible cases for the `salt`. In a first step, both `tokenID` and `salt` are converted to integers of precision higher than 256-bits and the appropriate multiples of the modulus are added. If the result does not overflow 256-bits, the algorithm must compute the associated commit and compare it to the correct on-chain commit of the note. The correct decryption of the commit openings are the ones that produce the same result as the on-chain commit.

An alternative remediation would be to encode the top 8-bits of `token_id`, `token_value`, and `salt` together with the 160-bit `ercAddr` in one field element and then to encode the remaining lower 248-bits from `token_id`, `token_value`, and `salt` as three additional field elements (see also Issue M). This makes the conversion between `u32[8]` and `Fp` unique.

Status

The Polygon Technology team has replaced the encryption algorithm with a KEM-DEM construction along with an appropriate encoding to prevent the problem of the ciphertext being possibly decrypted into the wrong commit openings.

Verification

Resolved.

Suggestions

Suggestion 1: Remove Check on Ask in Double Transfer [Circuit]

Location

`/nightfall-deployer/circuits/double_transfer.zok`

Synopsis

Double transfer combines two commitments and allows the user to spend any value up to the sum of the two. By taking the `pkd` from `commitment[0]` for computing the hash of both commitments, the circuit constraints necessitate that the two commitments be based on the same public address `pkd` as well as private keys (`ask`, `nsk`, `ivk`). The check on the equality of the two keys `ask[0]` and `ask[1]` in line 70 is redundant since it is already implicitly enforced via the check on the commitment hashes.

Mitigation

We recommend that the redundant check be removed to avoid confusion and to simplify the code. Alternatively, the implementation of the recommendations in Issue N would remediate this suggestion.

Status
The Polygon Technology team has removed the redundant code.

Verification
Resolved.

Suggestion 2: Expand Test Coverage [All Components]

Location
[`/test`]

Synopsis

Nightfall 3 implements insufficient test coverage. Sufficient test coverage for success and failure cases helps to identify potential edge cases and protect against errors and bugs, which may lead to vulnerabilities or exploits. A test suite should include a minimum of unit tests and integration tests. End-to-end testing is also recommended to determine if the implementation behaves as intended.

Mitigation
We recommend that the test suite be improved to include tests for false challenges, redundant challenges, and challenge gas costs in the smart contracts component. For the wallet/client, we recommend implementing unit tests that cover success, failure, and edge cases.

Status
The Polygon Technology team has upgraded the existing test suite to include additional tests, which improves test coverage. However, the test coverage is not comprehensive.

Verification
Partially Resolved.

Suggestion 3: Document the Source of the MiMC Constants [Circuit]

Location
`/hashes/mimc/mimc-constants.zok`

Synopsis
The `MiMC` construction requires predefined constants that are chosen randomly with no algebraic relation between any two elements. In order for users to verify the absence of this relationship in the derivation algorithm, the constants must be documented.

Mitigation
According to the Polygon Technology team, the constants are selected based on the standard by the circom implementation, with the seed `MiMC` (6d696d63). We recommend adding the source of the `MiMC` constants and the derivation algorithm in the documentation.

Status
Since `MiMC` will be replaced by `Poseidon`, this suggestion is redundant.

Verification
Resolved.

Suggestion 4: Remove Unused and Commented-Out Code [Smart Contracts]

Location

Non-exhaustive:

`/nightfall-deployer/contracts/MerkleTree_Stateless.sol#L105-L106`

`/nightfall-deployer/contracts/MerkleTree_Stateless.sol#L258-L269`

`/nightfall-browser/services/transfer.js#L236-L260`

Synopsis

There are some instances of commented-out and unused code in the codebase. This reduces readability and can confuse reviewers and maintainers.

Mitigation

We recommend that unused and commented-out code be removed to make the smart contracts and other parts of the codebase more readable.

Status

At the time of the verification, the implementation of the recommendations remains pending.

Verification

Unresolved.

Suggestion 5: Avoid Code Duplication [Wallet/Client]

Location

Non-exhaustive:

`/crypto/encryption/modular-division.js#L37`

`/utils/crypto/modular-division.js#L37`

Synopsis

Parts of the Nightfall 3 client and wallet are duplicates of each other. In some cases, the copies are exact, and in others, very small differences exist between the otherwise identical source files. An example of this is within the wallet, where there is a duplicate implementation of modular division that differs slightly.

Code duplication carries the risk of subtle divergence where bug fixes or other changes are made to one copy and not the other.

Mitigation

We recommend that shared code be abstracted out in one location in order to avoid changes that only affect one copy of the code when it should affect both.

Status

The Polygon Technology team has stated that the two codebases will be extracted into their own repositories at a later date. This, however, will not resolve the problem. The reason for the code duplication stated by the Polygon Technology team is that React does not allow imports outside the `src` folder. We suggest extracting the duplicate code, especially if it concerns mathematical and cryptographical operations, into another repository. This way, either git submodules can be utilized to integrate the repository in multiple places, or it can be imported as a package.

Verification

Unresolved.

Suggestion 6: Update Naming of Keys [Wallet/Client]

Location

`/doc/in-band-secret-distribution.md`

`/nightfall-client/src/services/keys.mjs`

Synopsis

Currently, the naming of the keys was inspired by Zcash Sapling. However, the respective naming of the used keys does not accurately reflect their role and can result, first, in confusion for future secure development and, second, in assumptions on false security of the key functionality.

For example, we noticed that `pkd` is not, in fact, a diversified public key, as constructed in Zcash. Also, `ivk` is not being used as an incoming viewing key.

Mitigation

We recommend updating the naming of the keys, especially `pkd`, `ask`, `nsk`, and `ivk`, to accurately represent their functionality.

This suggestion is related to Issue N. We strongly suggest updating the key schedule in the zero-knowledge proof, in general, as described in Issue N. However, until then, we recommend updating the naming of the keys, as described in this suggestion.

Status

The Polygon Technology team has updated the naming of the keys as suggested.

Verification

Resolved.

Suggestion 7: Improve Error Handling in Wallet [Wallet/Client]

Location

Non-exhaustive:

`/wallet/src/utils/lib/key-storage.ts`

Synopsis

In various locations within the wallet code, errors are not handled properly. In some locations, the lack of a connection to MetaMask is not covered, and a missing connection will cause an error (UI crashing with stacktrace).

Mitigation
We recommend that all potential states of the wallet be considered in error handling to include missing connections to MetaMask or other sources as well as missing connections to other resources.

Status
The Polygon Technology team has stated that they intend to implement the recommended mitigation in the future. Hence, the issue remains unresolved at the time of verification.

Verification
Unresolved.

Suggestion 8: Allow Users to Create More Nightfall Accounts [Wallet/Client]

Location
`/main/wallet/src`

Synopsis
Currently, the number of Nightfall 3 accounts for a user is fixed to a number supplied at the build-time of the wallet server and is set to 10 by default. Users should not be limited in the number of accounts they want to create. This enhances privacy options, as users can use different accounts for new transactions or new recipients.

Mitigation
We recommend implementing the ability for users to create more accounts.

Status
The Polygon Technology team has stated that they intend to implement the recommended mitigation in the future. Hence, the issue remains unresolved at the time of verification.

Verification
Unresolved.

Suggestion 9: Unify Number Representations [Wallet/Client]

Location
`/main/wallet/src`

`/main/nightfall-client/src`

Synopsis
Numbers, especially cryptographic keys and encrypted values, are represented in inconsistent formats in the code, including `BigInt` values, hexadecimal strings (sometimes but not always prefixed with “`0x`”) and “general numbers” (GN). This complicates and confuses the readability of security-sensitive parts of the code.

Mitigation
We recommend using one number format suitable for each task (see also Issue X). There should be clear separations where data enters and exits the system. It should be converted to internal representation

once on entry and to any required external representation once on exit, with no unnecessary conversions in between.

Status

The Polygon Technology team has stated that they intend to implement the recommended mitigation in the future. Hence, the issue remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 10: Separate Configuration from Constants [Wallet/Client]

Location

`/config/default.js`

Synopsis

The "default" configuration file mixes user-modifiable settings with constant definitions that should not be changed, such as ZERO and elliptic curve parameters.

Mitigation

We recommend moving immutable constants to a separate file.

Status

At the time of the verification, the implementation of the recommendations remains pending.

Verification

Unresolved.

Suggestion 11: Improve Nightfall 3 Documentation and Code Comments [All Components]

Location

`doc/`

Synopsis

The availability of comprehensive documentation outlining the system and interactions between the components is helpful for reviewers and end users who want to learn more about Nightfall 3, in addition to developers integrating it into their projects.

The existing documentation explains at a high-level how the different system components interact by providing details on the general architecture and some of the technical concerns. We found this information accurate and helpful. However, the documentation is missing more nuanced details on the implementation of each of the components and how components are expected to communicate with the wallet.

In addition, code comments are the most basic form of documentation and are critical for reasoning about the security of the implementation. Although there are code comments in many areas in the codebase, there are areas where code comment coverage is insufficient, which could lead to confusion and to vulnerabilities being missed.

The smart contracts component documentation was insufficient in that it did not describe all the functionality performed by the contracts and the interaction of the smart contracts with all the components in the system.

The documentation of the wallet/client implementations include descriptions of some HTTP endpoints, but several others that perform important functionality relating to commitments and key generation do not have such descriptions. The wallet and client implement some code comments. However, the code relating to cryptography and mathematics should have comprehensive comments.

Mitigation

We recommend the following improvements to the project documentation:

- Expanding the high-level system design documentation so that it is user-friendly and provides more specific details about the intended functionality of all wallet components, in addition to the wallet's interaction with all the external components; and
- Introducing a best practice section on how to use the wallet correctly to ensure users' privacy is not revealed due to wrong usage.

Smart Contracts

We recommend the improvement of the descriptions in the Nightfall 3 smart contracts documentation regarding the following:

- Transaction format and validation rules;
- Challenge conditions;
- The Optimist (proposer, challenger);
- Tests; and
- Trust and censorship assumptions necessitated by a single proposer and challenger.

In addition, we recommend the improvement of code comments in the smart contracts to include NatSpec documentations of all functions and other explanatory comments.

zk-SNARK Circuits

See Suggestion 3.

Wallet/Client

We recommend that the documentation be updated to include descriptions of the following endpoints:

- `GET  /commitment/…`
- `POST /finalise-withdrawal`
- `POST /generate-keys`
- `POST /set-instant-withdrawal`
- `POST /valid-withdrawal`

We recommend that the code comments in the wallet and client implementation be improved to describe parameters, general functionality, and output values for all functions, in addition to describing the functionality of all important lines of code.

Status

At the time of the verification, the implementation of the recommendations remains pending.

Verification

Unresolved.

Suggestion 12: Rewrite Withdrawal Function Making Notes Larger than 64-bits Withdrawable [All Components]

Location

`/nightfall-deployer/contracts/Structures.sol#L36`

`/nightfall-deployer/contracts/Shield.sol#L255`

`/nightfall-browser/services/commitment-storage.js#L525`

`/nightfall-deployer/circuits/double_transfer.zok#L88`

Synopsis

The field `value` in commitments is restricted to 64-bits for deposit and withdrawal by the functionality of the associated smart contracts. It is possible to create two commitments, with one having a value of more than 64-bit length and another having a value of less than 64-bit length in a double transfer transaction, from two commitments with values represented with 64-bit length. The newly created commitment with a value of more than 64-bit length cannot be withdrawn during the withdraw transaction.

(The Polygon Technology team mentioned that the code was updated to have 112-bit length values instead of 64-bit length values as of the commit provided for this report. However, this does not change the particularities of this suggestion.)

Impact

This results in un-withdrawable commitments with values of more than 64-bit length.

(In case the value length is increased again (e.g., to 223-bit), a commitment with a value of more than 224-bit length can be created through a double transfer transaction. This would lock the commitment of a value of more than 224-bit length since that commitment cannot be used as an input commitment for a double transfer transaction to be split up again into smaller valued commitments. This scenario is unlikely at the moment since with the current 64-bit constraint for deposit on the `field` value, too many transactions are necessary to generate a 224-bit commitment. However, in case the Polygon Technology team decides to enable larger deposits, this scenario becomes feasible.)

Preconditions

A commitment with a value of more than 64-bit length needs to be created in a double transfer transaction. The commitment needs to be tried for a withdraw transaction.

Technical Details

Two commitments of values represented in 64-bit length are being used as input for a double transfer transaction, with the intention of producing one commitment with a value greater than 64-bit length and another commitment with a value less than 64-bit length.

When a commitment with a value greater than the 64-bit length is processed in a withdraw transaction, the transaction fails in `Shield.sol`.

Remediation

We recommend adding constraints to enforce that values in input and output commitments adhere to the 64-bit length in all of the deposit, single transfer, double transfer, and withdraw circuits.

Status

At the time of the verification, the implementation of the recommendations remains pending.. However, a resolution is being discussed internally by the Polygon Technology team. Discussions include adding two circuits until this problem is mitigated: A withdraw with a change circuit and a single input with a double output transfer circuit. This would help address the underlying problem behind this suggestion.

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 multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, and zero-knowledge protocols. 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. We are an international team that believes 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 include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As 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 and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and 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 before having 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 and comprehensive 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 Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, 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 a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.