Least Authority
PRIVACY MATTERS

Mobile IrisCode Self-Custody Upgrade

Security Audit Report

Worldcoin

Final Audit Report: 2 January 2025

Table of Contents

Overview

Background

Tools for Humanity Corporation has requested that Least Authority perform a security audit of their Mobile IrisCode Self-Custody Upgrade project, which allows users to self-host biometric data on their personal device while supporting authentication for the World ID service.

Project Dates

- **October 7, 2024 - November 29, 2024:** Initial Code Review *(Completed)*
- **December 3, 2024:** Delivery of Initial Audit Report *(Completed)*
- **January 2, 2025:** Verification Review *(Completed)*
- **January 2, 2025:** Delivery of Final Audit Report *(Completed)*

The dates for verification and delivery of the Final Audit Report will be determined upon notification from the Worldcoin team that the code is ready for verification.

Review Team

- George Gkitsas, Security / Cryptography Researcher and Engineer
- Jasper Hepp, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Mobile IrisCode Self-Custody Upgrade project followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

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

- GitHub Repository:
  https://github.com/Modulus-Labs/Remainder

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

- `7be737bae0e1ac958967162224d6a6dd6dafea63`

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

- Modulus-Labs/Remainder:
  https://github.com/LeastAuthority/modulus-labs-remainder/tree/Phase2

For the verification, we examined the Git revision:

- `24ed455ecf724f8d7581b3e50f1bd710fe07dc37`

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:

- S. Papini and U. Haböck, "Improving logarithmic derivative lookups using GKR." *IACR Cryptology ePrint Archive*, 2023, [PH23]
- J. Thaler, "Proofs, Arguments, and Zero-Knowledge." *Georgetown University*, 2023, [Thaler23]
- J. Thaler, "Time-Optimal Interactive Proofs for Circuit Evaluation." *IACR Cryptology ePrint Archive,* 2013, [Thaler13]
- R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zkSNARKs without trusted setup" (referred to as the Hyrax paper). *IACR Cryptology ePrint Archive*, 2017, [WTS+17]
- "Scaling Intelligence: Verifiable Decision Forest Inference with Remainder." *Modulus Labs*, 2024. [Modulus Labs 24]
- Project Overview: https://future-seaplane-bec.notion.site/Mobile-IrisCode-Client-side-ZKML-for-Private-Biometric-Authentication-efbc6ba76206444fbea91e2f9ffdab23#a2a3c1e026da4564bd40c1d06cfdf089
- Worldcoin Website: https://worldcoin.org
- Tools For Humanity Website: https://www.toolsforhumanity.com
- Constraining for the "response zero" case using the "complementary representation" (ext): https://future-seaplane-bec.notion.site/Constraining-for-the-response-zero-case-using-the-complementary-representation-ext-9f16a435a6cf4f87a3572923ec7e7adb
- Relevant Files (code + circuit description): https://future-seaplane-bec.notion.site/Audit-Companion-IrisCode-ZK-Circuit-Hyrax-Prover-Verifier-ext-bacfc8c8f24a417aaaf0f48dee93783b#7836bf11e1cb4b5c93955956a93a8631
- Hyrax IP: The Super Detailed List (ext): https://future-seaplane-bec.notion.site/Hyrax-IP-The-Super-Detailed-List-ext-88c439290df74123976c611b70ac812f
- Circuit description (v2) (ext): https://future-seaplane-bec.notion.site/Circuit-description-v2-ext-c907a39efc314338aafa5d565c91c41c
- LogUp (ext): https://future-seaplane-bec.notion.site/LogUp-ext-f846956acc3640a68bad51f7897fe32f
- Building Babylon | Understanding LogUp: A Royal Road: https://building-babylon.net/2024/02/14/a-royal-road-to-logup
- Proving Iris Code Calculation | Enabling Model Upgrades Under Self-Custody | Interim Progress Report (Google Doc Presentation): https://docs.google.com/presentation/d/1eoX1-wy7TRqRtbarNVA31wBeF73mPP1PxhCIBaPtD8c/edit#slide=id.g2a710c97050_2_1492

In addition, this audit report references the following documents:

- E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model." *IACR Cryptology ePrint Archive*, 2004, [BCO04]
- A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems." *Springer*, 1986, [FS86]
- N. Thoi Minh Quan, "00." *cryptotosubtlety*, 2021, [Quan21]
- Crate std: https://doc.rust-lang.org/std
- To panic! or Not to panic!: https://doc.rust-lang.org/book/ch09-03-to-panic-or-not-to-panic.html
- Crate zeroize: https://docs.rs/zeroize/latest/zeroize/index.html#

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and alignment with specifications;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Vulnerabilities in the code leading to adversarial actions and other attacks;
- Protection against malicious attacks and other methods of exploitation;
- Inappropriate permissions and excess authority;
- Data privacy, including data leaking, unexpected data inference, voids, and information integrity;
- Any applicable use of cryptography, including key management and encryption protocols;
- Challenges to transparency and governance measures; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Modulus Labs is implementing a zk-SNARK based on the Hyrax protocol [WTS+17] and a circuit to prove the correct upgrading of a Worldcoin user in zero-knowledge. This provides Worldcoin with a tool to upgrade its users to a newer version of the iris code without revealing sensitive data to the Worldcoin server by executing the privacy-sensitive computations on the user's phone.

The setup is as follows: the orb sends an image of the iris to the user. The user generates a proof on the mobile phone. The Worldcoin server verifies the obtained proof. The image data from the user, which is obtained from the orb, is not revealed to the Worldcoin server. In its current version, the iris code is shared with the Worldcoin server, but there are plans to also hide the iris code from the Worldcoin server. The proof consists of six Hyrax proofs for six different circuits (version 2 and 3, `is_mask` on and off, and for the left and right eye) and proves the correct transition from version 2 to version 3 in zero-knowledge.

System Design

Our team examined the design of the Mobile IrisCode Self-Custody Upgrade project and found that security has generally been taken into consideration as demonstrated by the usage of well-designed and implemented cryptography and thorough testing. Furthermore, the usage of the memory-safe language Rust adds to the project's security.

Circuit

We reviewed the circuit implemented here, as compiled inside the test function `test_upgrade_v2_v3`. The circuit is proving the correct upgrade of an existing Worldcoin user from version 2 to version 3 (documented here and here). The circuit takes as private input the image data as well as the digits and their multiplicities for the lookup constraints. The iris code is a public input. The circuit proves that the iris code and iris image match. The circuit description is generated by the verifier independently of the prover. The circuit is compiled in the folder `remainder_prover` (partly documented in the papers [Modulus Labs 24] and [Thaler13]) into a representation that can be used as an input to the Hyrax protocol.

We did not identify any missing constraints in the circuit.

For performance reasons on the prover side, the circuit does not enforce the correct padding of data inputs. Instead, the verifier checks the correct padding for the iris code outside of the circuit. The iris image data is not checked for correct padding; however, the image commitment and a hash on this commitment bind the prover to the image. The verifier can trust the image commitment since it is obtained from the trusted orb. Additionally, the public values are verified outside of the circuit against the expected values. We did not identify any issues in this regard.

Hyrax Protocol

We reviewed the correctness of the prover and the verifier implementation. These are based on the Hyrax protocol ([WTS+17]) with extensions from [Thaler13] for structured circuits (documented here and here) as well as lookup arguments based on [PH23] (documented here and here). The protocol makes extensive use of the Sumcheck protocol as well as Pedersen commitments ([Thaler23]) and is implemented over the `BN256` elliptic curve.

We found a critical soundness issue. The verifier does not check the shape of the proof data obtained from the prover. A malicious prover can omit parts of the proof, effectively only proving parts of the circuit (Issue A).

We found a missing verifier check on the length of the commitments on the private inputs. While we did not identify an exploit based on this issue for the current Worldcoin setup in scope, it could nevertheless lead to a critical soundness issue in other settings and circuits (Issue B).

We reviewed the correct implementation of the Sumcheck protocol for the following layers that are used in the Worldcoin circuit: regular layer, identity layer, and matrix multiplication layer (`MatMul`). The Sumcheck protocol has been adjusted, such that the final checks are reduced to one by "squashing" the provers' messages. We found that the proof of products on X, Y, Z occurs at the end of the subprotocol here, even though Figure 1 in the Hyrax paper would expect it before the call to `ProofOfSumcheck::prove`. We did not identify any security-relevant issue with respect to this minor deviation and with the implementation of the Sumcheck protocol in general.

We found a difference between the processing of the input layer in the code versus the processing described in the Hyrax protocol. We concluded that this deviation does not pose any security risks. Nevertheless, our team noted that it adds unnecessary proof data and hence also adds inefficiencies. We recommend adjusting this part and following the Hyrax paper (Suggestion 3).

We reviewed the correct implementation of the Fiat-Shamir transformation, which is implemented as a Sponge construction with the Poseidon hash function over the elliptic curve. We recommend removing unnecessary data from the transcript and adding a domain separator (Suggestion 2). We did not identify any security-relevant issues related to the Fiat-Shamir transformation.

We reviewed the lookup argument ([PH23]) and found that the final check is missing. The Worldcoin team stated that they chose not to include this check, as they found it to be unnecessary due to an argument involving the Schwartz-Zippel lemma. We recommend documenting and justifying this design decision (see also Suggestion 5). We did not identify any additional concerns in this regard.

Zero-Knowledge

The Hyrax proof system utilizes zero-knowledge. We reviewed the implementation of the prover to ensure that it does not reveal any data to the verifier besides the proof.

We reviewed the correct implementation of the Pedersen commitments. The generators are constructed by the verifier from a constant and publicly known string. We did not identify any issues related to these.

All Pedersen commitments have the hiding property by adding a blinding factor. Only for the commitments on the Fiat-Shamir claims, and on the claims on the public values, the blinding factors are shared with the verifier, which is necessary for correct verification. The commitment that is part of the evaluation proof of an input layer proof is also shared with the blinding factors. This is not a security-relevant issue; nevertheless, we recommend removing this commitment (Suggestion 3). Overall, we did not identify any issues.

Timing Attacks

In addition, we considered timing attacks focusing on leakage coming from the elliptic curve scalar multiplication operation. Two non-constant time implementations are used, the first being implemented by the Worldcoin team (`scalar_mult`), and the second being part of the `halo2curves` dependency.

We did not identify any security-relevant values that can leak through the `halo2curves` scalar multiplication. We found that the iris image is an asset that leaks through `scalar_mult`, which is relevant only if the library is used without the pre-commit functionality.

We did not find a way for passive attackers to exploit the system. On the other hand, an active attacker who can invoke the target code on demand can utilize correlation timing attacks ([BCO04]), which enables the deduction of the full values of the scalars used. This requires the attacker to be logically local. An attacker with these capabilities would most probably have other less demanding attack paths available. Due to this reason, as well as the steep attack requirements, our team concluded that the aforementioned attack does not pose an immediate risk. However, we still recommend assessing constant-time implementations as a precautionary step (Issue D).

Other Areas of Investigation

Our team considered the zeroization of sensitive values, such as iris data and blinding factors. This is a precaution to counter any information leakage coming from residual data (Issue E).

We assessed susceptibility in Denial-of-Service (DoS) attacks. We did not identify any areas of memory or execution time blow-ups. We assessed the usage of panic-inducing logic (e.g., assertions) as an attack vector for causing DoS through forcing application crashes. We concluded that the code snapshot under audit is not susceptible to this attack vector since each request is handled in a single thread. However, if the library is integrated in a way where multiple requests (or aspects of them) are handled within the same thread, this could enable DoS attacks. As a precaution and future-proofing, we recommend replacing panic logic with error management in code paths where untrusted input can cause a panic (Suggestion 6).

We assessed attacks based on proof re-using. We identified an issue where the left and right iris proofs are not distinguished and, thus, a prover can pass the verifier checks by presenting a proof of only one iris side twice, for both the left and right eye. For further details, see Issue C.

We explored attacks based on the paper [Quan21]. We performed several modifications of the prover in an attempt to break soundness using a combination of values of zeroes and ones in the prover's inputs to the verifier. We did not identify any issues.

Code Quality

We performed a manual review of the repositories in scope and found that the project is generally well-organized and utilizes Rust's abstractions. Additionally, the naming is clear when combined with the context of the relevant documentation. However, due to ongoing development, there are unused and/or obsolete code artifacts, which warrant further refactoring. We also identified some instances of code duplication and minor logical errors, which we recommend be addressed (Suggestion 5).

Tests

The Worldcoin team has implemented sufficient tests, which showed high coverage of important logic.

Documentation and Code Comments

The project documentation provided by the Worldcoin team and the code comments sufficiently describe most of the intended functionality of the system. The Worldcoin team was also responsive and helpful in answering our questions. However, our team noted that the project documentation is scattered across different documents and resources. We suggest creating one document that comprehensively describes the protocol. In addition, we found an undocumented deviation from the log-up protocol and we suggest documenting it (Suggestion 5).

Scope

The scope of this review included the `remainder_hyrax` folder and all the functions that are called from within this folder into the other parts of the codebase, the most relevant of which are the Worldcoin circuit logic and compilation, the Sumcheck logic for each of the layers in scope (`regular`, `identity`, and `MatMul`) and the Fiat-Shamir elliptic curve transcript with the Poseidon Hash function. As top-level functions, we considered the logic in the test file `upgrade.rs` and the test function `test_upgrade_v2_v3` as our starting point to review the Hyrax protocol, the circuit, and its integration into the Worldcoin system. In addition, we reviewed the folder `remainder_shared_types`.

Our team did not review the data parallel option that is only partially implemented and is currently not used by the circuit in scope nor did we investigate any other function that had not been called by the core Hyrax protocol implemented in `remainder_hyrax`. The implementation of Ligero in the folder `remainder_ligero` was also excluded from the scope of this audit.

Note that since the integration and deployment configuration was out of the scope of this audit, we were unable to further research and validate, or dismiss, concerns about the security during the data-in-use lifecycle phase of the iris image and mask data. We therefore recommend that upstream users and integrators further assess how to protect this data when it is processed by this library, based on the system-level threat model and attack surface.

Dependencies

We examined the dependencies implemented in the codebase and found that the project generally relies on up-to-date and secure dependencies, with the exception of one indirect dependency that is reported as unmaintained, as described in Suggestion 1.

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: Missing Verifier Check on the Shape and Completeness of the Proof Data	Resolved
Issue B: Missing Verifier Check on Length of Input Commitment	Resolved
Issue C: No Distinction Between Left and Right Irises	Resolved

Issue D: Timing Side-Channel Leakage on EC Scalar Multiplication	Resolved
Issue E: Leakage Through Residual Data	Resolved
Suggestion 1: Upgrade or Replace Unmaintained Dependencies	Resolved
Suggestion 2: Adjust Fiat-Shamir Transcript	Resolved
Suggestion 3: Remove Unnecessary Step from the Input Layer	Resolved
Suggestion 4: Improve Code Quality	Partially Resolved
Suggestion 5: Improve Documentation for the Lookup Constraints	Resolved
Suggestion 6: Eliminate Panics Caused by Untrusted Input	Resolved

Issue A: Missing Verifier Check on the Shape and Completeness of the Proof Data

Location

`remainder_hyrax/src/hyrax_gkr/mod.rs`

Synopsis

The verifier does not check the shape of the proof data with respect to the circuit description. The prover can thus omit certain parts of the proof, which results in a soundness issue.

Impact

Critical. A malicious prover can omit certain parts of the proof and the verifier would accept the incomplete proof as true.

Preconditions

The circuit needs to have a branch for which the claim reduction from output to input layer does not contain any Fiat-Shamir node. This is because of the way the Fiat-Shamir nodes are verified by the verifier.

The precondition is fulfilled for the Worldcoin circuit in scope; that is, the circuit contains branches that do not contain any Fiat-Shamir node.

Feasibility

High.

Technical Details

The verifier receives the proof from the prover and hence cannot trust that the proof is actually complete. The verifier generates the circuit description independently of the prover and should therefore check the shape of the proof against the trusted circuit description.

Because of the way the verifier checks the Fiat-Shamir nodes, these cannot be dropped by an attacker from the proof (see Precondition section above).

For an attack scenario against the Worldcoin circuit, the prover can drop, for example, everything that is related to the binary check on the iris code; that is, any proof data related to the `OutputNode(zero)` node and the `BitsAreBinary` component as well as the corresponding claim for the iris code layer contained in the `claim_on_public_values`. Due to this, the attack skips the binary range check against the iris code within the proof verification.

Remediation

We recommend adding checks in the verifier code that compare the circuit description against the proof struct. In particular, we recommend that the verifier add the following checks between the structs `GKRCircuitDescription` and `HyraxProof`:

- The number of the Fiat-Shamir challenges against the Fiat-Shamir claims;
- The number of input layers against the number of claims on public values added to the number of Hyrax input proofs;
- The number of intermediate layers against the number of layer proofs; and
- The number of output layers against the number of output layer proofs.

It is not necessary to check that the number of public inputs in the proof struct matches the number of claims on the public values since this is checked implicitly through the `input_layer_claims`.

Status

The required checks have been implemented by the Modulus Labs team.

Verification

Resolved.

Issue B: Missing Verifier Check on Length of Input Commitment

Location

[`src/hyrax_pcs/mod.rs#L359`]

Synopsis

The verifier does not check the length of the commitment vector `T'` against trustworthy data that is not obtained from the prover. An attacker could drop parts of the private inputs and hence skip parts of the circuit logic for the dropped input. The Worldcoin setup in its current state is not affected.

Impact

For the current setting of the Worldcoin circuit, the impact is low. Outside of this given setting, the impact is critical. An attacker could potentially drop parts of the private input commitments. The impact on the Worldcoin setup appears negligible given that the verifier checks the input commitment on the iris image with trusted data from the Orb outside of the circuit.

Preconditions

If the input commitments to the circuit are verified outside of the circuit, an attack is not possible.

Feasibility

For the current setting of the Worldcoin circuit, the feasibility is low. Outside of this given setting, the feasibility is high.

Technical Details

In the Hyrax paper ([WTS+17]), a subprotocol (Section 6) describes how to reduce communication costs around the verification of the witness data. The core idea is to reduce the verification via a polynomial commitment scheme for multilinear polynomials.

More specifically, for a witness `w` (viewed as a matrix) and corresponding commitments `T_1 … T_k` over the rows, the paper introduces the multi-commitment `T'` constructed from the commitments `T_i`. In particular, the verifier constructs `T'` from the helper vector `L` and the commitments `T_1…T_k`.

In this construction, the code does not check that the number of commitments `T_1…T_k` (obtained from the prover) actually matches the length of the `L` vector (that can be trusted because it is constructed independently from the prover). The code only checks if the number of commitments is correct if the vector `L` is empty and there is only one commitment `T_1`.

In the setting of the Worldcoin circuit, the input commitments are on the iris image and the digits value. A malicious prover could potentially drop parts of the vector of commitments to the iris image and the corresponding parts of the digits values. This would allow the attacker to commit only to parts of the iris image bytes string and, hence, to skip parts of the verification between the iris image and the iris code. However, the commitment on the iris image is generated by the (trusted) Orb as a precommitment, and the verifier is comparing a hash on the commitment with the input commitment from the prover. Hence, in the current setting, our team did not identify any immediate attack.

This attack is nevertheless feasible for other circuits. In particular, if the Worldcoin circuit is modified (for example, if the iris code is made a private input), the sketched attack would also become feasible for this setting.

Remediation

We recommend adding a check to verify that the length of the vector `L` equals the number of rows in the input commitment matrix.

Status

The required check has been implemented by the Modulus Labs team.

Verification

Resolved.

Issue C: No Distinction Between Left and Right Irises

Location

`src/hyrax_worldcoin/test_worldcoin.rs#L115`

Synopsis

The prover can pass the verifier check by admitting only one iris proof for both the left and right sides.

Impact

This results in weaker authentication assurances than intended.

Preconditions

This issue can occur if the attacker has access to one valid iris proof or access to any two valid iris proofs.

Feasibility

For this issue to become an attack, the attacker must first obtain one valid iris proof.

Technical Details

The verifier expects proofs for both the left and right irises. Both proofs are checked using the same circuit (for each version). There is no distinction between the circuit invocations for the left and right irises, which allows using a duplicate of a valid iris proof.

Remediation

We recommend introducing a mechanism that assures that each provided proof corresponds to the expected iris side.

Status

In the proposed solution, the prover sends signed hashes of each of the commitments. The verifier then checks the signatures of those hashes and compares the hash values of the received commitments against them.

We find the proposed solution adequate. The current implementation lacks signature checking, which is outsourced to the backend due to technical obstacles. We assume the correctness and secure implementation of the signature checking and thus consider this issue resolved.

Verification

Resolved.

Issue D: Timing Side-Channel Leakage on EC Scalar Multiplication

Location

`src/curves/mod.rs#L432`

Synopsis

Elliptic curve scalar multiplication is non-time-constant, which can leak sensitive information through a timing side-channel.

Impact

Critical. This issue can lead to the full disclosure of an iris image. Note that the attack is not easily scalable due to the targets operating within diverse systems and also because each target needs to be attacked individually.

Preconditions

The code would have to be used without pre-commits. The attacker needs to be logically local to the device and able to cause repeated executions of the target code.

Feasibility

The feasibility depends on the application-level design decisions and threat model. Due to this reason, our team did not have enough information to accurately assess feasibility.

Technical Details

Elliptic curve scalar multiplication is implemented using the double-and-add algorithm in the function `scalar_mult`. Double-and-add is susceptible to side-channel leakage. `scalar_mult` is used by the prover during the iris image commitment calculations and thus leaks information about the iris image. There is no leakage of the iris image if pre-commits are used, and any other leakage occurring during the calculation of other commitments is deemed unimportant to an attacker.

Our team did not find a way for passive attackers to exploit the system, as they are unable to obtain a significant amount of information. A passive attacker can only obtain timing measurements when the user decides to use the application. Additionally, since a user will only invoke the upgrade once (or just a handful of times in case of errors), a passive attacker can acquire only one timing measurement. With this measurement, the attacker can, in the theoretically ideal scenario, infer information about the number of leading zeros and the total amount of zeroes/ones present in the scalar. This information is not meaningful enough for an attacker in this context. Moreover, given that in a realistic scenario there exist measurement inaccuracies and unpredictable timing variations, we consider this attack vector fully ineffective.

An active attacker who can invoke the target code on demand can obtain multiple timing traces and thus utilize correlation timing attacks ([BCO04]). Correlation attacks can lead to full disclosure of the actual scalar value. Any obstacles added by noise in measurement traces can be overcome by obtaining a larger amount of traces and utilizing more advanced techniques (higher-order attacks, template attacks, etc.). This is therefore a potent attack vector that results in the disclosure of valuable information.

This attack vector can be exploitable if the attacker is able to invoke the target on demand, which in turn requires the attacker to be logically local and have sufficient authorization. Furthermore, any needed user interaction, such as active action authorization, can be a blocker to this attack. While we did not have enough information to assess the full attack path, our team noted that the prerequisites are demanding and satisfying them is highly likely to lead to other preferable attack paths. We thus conclude that while the impact of this attack is high, the likelihood is very low, but still existent.

Remediation

We recommend re-implementing `scalar_mult` using constant-time scalar multiplication algorithms.

Status

The Modulus Labs team has re-implemented the scalar multiplication to run in constant time.

Verification

Resolved.

Issue E: Leakage Through Residual Data

Location

`src/hyrax_worldcoin/upgrade.rs#L260`

Synopsis

Sensitive data could reside after the end of the application's execution. An attacker could be able to access them, leading to their disclosure.

Impact

This issue could potentially result in the full disclosure of the iris image data.

Preconditions

The attacker would have to be logically local to the target device.

Feasibility

The feasibility depends on the application-level design decisions and threat model. Due to this reason, our team did not have enough information to accurately assess feasibility.

Technical Details

Residual data can leak information about sensitive values, such as blindings and iris data. There exist multiple ways for attackers to obtain residual data, such as by allocating memory recently freed by the target application, causing core dumps, or via swap memory.

Remediation

We recommend utilizing memory zeroization, such as https://docs.rs/zeroize/latest/zeroize/index.html#, at the end of the lifecycle of sensitive values in order to reduce the leakage surface.

Status

The `zeroize` library is now utilized for all blinding factors and other sensitive randomly generated values.

Verification

Resolved.

Suggestions

Suggestion 1: Upgrade or Replace Unmaintained Dependencies

Synopsis

Running `cargo deny check advisories` on the codebase reveals that the dependency `derivative` is unmaintained. The relevant advisory is https://rustsec.org/advisories/RUSTSEC-2024-0388.

Mitigation

We recommend updating or replacing the reported dependency.

Status

The unmaintained dependency was removed by upgrading the direct dependency `ark-std` to a version that uses educe instead of `derivative`. `cargo deny` still reports issues with `derivative` coming from `remainder-ligero`. Since `remainder-ligero` is out of scope, we consider this issue fully resolved.

Verification

Resolved.

Suggestion 2: Adjust Fiat-Shamir Transcript

Location

[`src/hyrax_gkr/mod.rs#L96`](src/hyrax_gkr/mod.rs#L96)

[`src/hyrax_gkr/mod.rs#L231`](src/hyrax_gkr/mod.rs#L231) `src/hyrax_pcs/mod.rs#L313`

`src/hyrax_pcs/mod.rs#L372`

Synopsis
We recommend adjusting the Fiat-Shamir transcript as follows:

1. The elements that are absorbed first in the Fiat-Shamir transcript are currently the public inputs. We recommend adding a domain separator before absorbing the relevant elements from the protocol.
2. The commitment `T’` relevant for the subprotocol `Proof-of-Dot-Product` is currently absorbed in the transcript. This is unnecessary since the verifier constructs this independently from data it can trust (the `l_vector` and the input commitments).

Mitigation
We recommend addressing the points listed above.

Status
The Modulus Labs team has introduced the domain separators as suggested. In addition, The `Proof-of-Dot-Product` has been removed.

Verification
Resolved.

Suggestion 3: Remove Unnecessary Step From the Input Layer

Location
`src/hyrax_pcs/mod.rs#L372`

Synopsis
The handling of the input layer in the prover and verifier code deviates from the specification described in the Hyrax paper ([WTS+17]). The code adds unnecessary additional steps that can be removed to make it more efficient.

From the verifier perspective, the code performs the following steps:

1. Starting in the `verify` function for the input layer here, the inputs are the claims from the previous layers, the layer description, and a proof for this input layer (as well as the Pedersen committer and the transcript).
2. In the first step, the claims on the previous layer are aggregated in L125.
3. In the second step, the evaluation proof is verified with the input commitment from the input layer proof and the evaluation point in the aggregated claim.
4. In the third and final step, the `Proof-of-Equality` between the commitment generated in the evaluation proof and the evaluation in the aggregated claim is verified.

The last step is unnecessary. Instead, the evaluation from the aggregated claim can be directly used as an input to the `verify` function of the evaluation proof. From the proof, the `Proof-of-Equality`, as well as the commitment to the evaluation in the evaluation proof, could both be dropped.

Note that our team did not identify any security-relevant issue with respect to this deviation.

Mitigation

We recommend implementing the suggested reduction in the handling of the final input layer.

Status

The unnecessary `Proof-of-Equality` has been removed.

Verification

Resolved.

Suggestion 4: Improve Code Quality

Synopsis

During our extensive review of the codebase, our team identified opportunities for improving the readability and maintainability of the codebase. Below we list some of our findings:

- The following check is redundant because it has already been checked at a lower level in `src/hyrax_worldcoin/test_worldcoin.rs#L119`.
- The following comment is inconsistent. In the code, the point at infinity is when MSB is 1: `src/curves/mod.rs#L291-L292`.
- The function `get_scalar_field_challenge` here does not have an error message and has undocumented assumptions about the elliptic curve implementation.
- Abstraction utilization can be improved. For example, `opportunity` can be abstracted out with `Self:identity` in:
    - `src/curves/mod.rs#L275-L276`
    - `src/curves/mod.rs#L347`
    - `src/curves/mod.rs#L381`
- In `src/digits/mod.rs#L52`, `ilog2` rounds down, but the correct calculation should round up (for example, `.log2().ceil()`), as it is meant to count the needed number of bits.
- Assertions can be replaced with error returns for the outcomes of protocol checks in the following locations (non-exhaustive list):
    - `hyrax_primitives/proof_of_dot_prod/mod.rs#L163-L164`
    - `hyrax_primitives/proof_of_dot_prod/mod.rs#L145-L146`
    - `hyrax_primitives/proof_of_product/mod.rs#L111-L114`
    - `hyrax_primitives/proof_of_opening/mod.rs#L75`
    - `hyrax_primitives/proof_of_equality/mod.rs#L72`
    - `hyrax_primitives/proof_of_dot_prod/mod.rs#L145-L146`

Mitigation

We recommend addressing the points listed above.

Status

All issues have been addressed except for replacing assertions with errors. The Modulus Labs team has stated that this will be addressed structurally and work is ongoing.

Verification

Partially Resolved.

Suggestion 5: Improve Documentation for the Lookup Constraints

Synopsis

Currently, the documentation on the lookup constraints (see here) does not explain why the final boundary check has been dropped. The final boundary condition checks whether the lookup constraint satisfies `(phi_0(0) = 0 and phi_1(0) != 0)`. According to the Worldcoin team, the security of dropping this check can be reduced to an argument around the Schwartz-Zippel Llemma.

Mitigation

We recommend adding the justification to the documentation on the lookup constraints.

Status

The Modulus team has created a soundness analysis. A reference to it has been added to the code..

Verification

Resolved.

Suggestion 6: Eliminate Panics Caused by Untrusted Input

Synopsis

A Denial-of-Service (DoS) attack can be caused by consistently forcing a server to crash. In Rust, panicking is a form of a crash, which abruptly ends the execution of the current thread. If a panic can occur by selecting specific values of an untrusted input, then depending on the responsibilities of the crashed thread, an attacker might be able to cause a DoS.

An example case is in `src/pedersen/mod.rs#L276`, where the prover controls the incoming commitment and can therefore force a `panic!` on the verifier.

The code snapshot under audit handles each prover request in its own thread. Consequently, causing a crash through panicking will only affect the malicious request. However, if the code gets integrated in a way where multiple prover requests (or aspects of them) are not handled in isolated threads, then an attacker might gain the ability to cause a DoS.

Mitigation

As a precaution, we recommend assessing all untrusted input code paths and replacing any panic-inducing checks with error propagation.

Status

The Modulus Labs team has addressed this suggestion by wrapping the entire verify function with a catch-all error handling.

Verification

Resolved.

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, zero-knowledge protocols, and consensus 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.