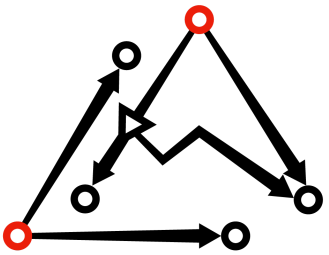

Least Authority

PRIVACY MATTERS

MPC Protocol for Uniqueness Check

Security Audit Report

Worldcoin

Final Audit Report: 4 April 2024

Table of Contents

Overview

Background

Worldcoin has requested that Least Authority perform a security audit of their MPC Protocol for the uniqueness check.

Project Dates

- **February 26, 2024 - March 22, 2024:** Initial Code Review *(Completed)*
- **March 26, 2024:** Delivery of Initial Audit Report *(Completed)*
- **April 4, 2024:** Verification Review *(Completed)*
- **April 4, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jasper Hepp, Security Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the MPC Protocol for the Uniqueness Check 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:

- MPC Uniqueness Check: https://github.com/worldcoin/mpc-uniqueness-check
- Signup Service: https://github.com/worldcoin/signup-service/tree/main/internal/mpc
  - folder in scope:
    - `internal/mpc` and `internal/inflight`
  - additional files in scope:
    - `internal/signup/signup.go`
    - `internal/uniqueness/encode.go`
- Signup Processor: https://github.com/worldcoin/signup-processor/tree/main/internal/mpc
  - folder in scope:
    - `internal/mpc`
  - additional file in scope:
    - `cmd/root.go`

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

- MPC Uniqueness Check: `197116671b1144912667f19e361b38916c20645b`
- Signup Service: `1f60f8fe16ed459b8d3b97b4da73e5658475613d`
- Signup Processor: `9af14ebb8754e7b535e9d30319691f457bc65ee7`

For the verification, we examined the Git revision:

- MPC Uniqueness Check: `5d6ab0a99c65b3fd493ea08873be46883acab15c`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- MPC Uniqueness Check:
  https://github.com/LeastAuthority/mpc-uniqueness-check
- Signup Service:
  https://github.com/LeastAuthority/signup-service
- Signup Processor:
  https://github.com/LeastAuthority/smpc-signup-processor

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:

- Specification:
  https://github.com/recmo/mpc-iris-code/blob/main/specification.ipynb
  - Commit: `a5a208e89fa2fb8c2df8ddf328b64c97d5893dce`
- Website:
  https://worldcoin.org
- Iris Recognition Inference System:
  https://worldcoin.org/blog/engineering/iris-recognition-inference-system
- PDF documents provided by the Worldcoin team:
  - `mpc-iris-code_Readme.md` at main *(shared with Least Authority via email on 2 February 2024)*
  - `Recmo_mpc-iris-code` *(shared with Least Authority via email on 2 February 2024)*
  - `(9+) MPC Design` *(shared with Least Authority via email on 2 February 2024)*

In addition, this audit report references the following documents:

- `cargo_audit`:
  https://docs.rs/cargo-audit/latest/cargo_audit/index.html
- `gosec`:
  https://github.com/securego/gosec
- `nancy`:
  https://github.com/sonatype-nexus-community/nancy

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Correct data synchronization across different databases;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of Worldcoin's secure multi-party computation protocol (MPC Protocol) for the uniqueness check of iris codes, which Worldcoin is introducing into its signup flow (for a summary, see the Worldcoin blog post on the Iris Recognition Inference System). The uniqueness check computation for a new iris code is split among two nodes. By separating the parties into two independent nodes, this improves user privacy — assuming each of the nodes store only shares of the iris code and report back the result of the uniqueness check to the signup service. As a result, neither the nodes nor a central authority has access to the full iris code. We reviewed the first version of the implementation that still lacks, for example, the encryption of the queries that contain shares of the iris code, which is necessary to fully achieve privacy enhancement.

We reviewed this implementation of the MPC Protocol for uniqueness check in the `mpc-uniqueness-check` for adherence to the current protocol specification as well as the integration of the MPC Protocol into the existing infrastructure, namely the signup service and the signup sequencer.

The specification is hand-crafted based on primitives and considerations from other protocols. In its current state, it has neither a formal specification nor a security proof. Since the specification itself was out of scope, we performed the audit under the assumption that the MPC Protocol design is secure and satisfies all required properties. As a result, we recommend a formal specification and a security proof be developed and audited in the future.

System Design

Our team examined the design of the MPC Protocol and found that security has generally been taken into consideration as demonstrated by the use of Rust, a robust, memory-safe language known for its efficiency, performance, and reliability.

The implementation is currently in the early stages of development. The queries are sent unencrypted, and only the shares (but not the masks) are stored encrypted. The code is not yet used in production, and its functionality is currently disabled in the accompanying repositories: signup service and signup sequencer.

The specification and the code assume a semi-honest setting. More specifically, the parties are assumed to execute the protocol faithfully and do not learn any information about the shares of the other parties.

We compared the codebase against the specification and did not find any issues, although we did not review the specification itself, as noted above. However, we identified an edge case (Suggestion 1) and some dependency issues (Issue A).

Our team also examined the integration into the existing codebase (both, in the signup service and signup sequencer). We could not identify any issue with the uniqueness check. Additionally, we found that a unique signup attempt is correctly added to the databases, and that the different databases are kept in sync. We did not find any issues in this area of our review.

Code Quality

We performed a manual review of the repositories in scope and found the code to be clean, well-organized, and of high quality, in that it adheres closely to development best practices.

Tests

The MPC Protocol includes test coverage. However, our team did not assess whether test coverage was sufficient, as the tests were out of the scope of this audit.

Documentation and Code Comments

The project documentation provided for this review was generally sufficient. Additionally, our team found that the codebase is well-commented and organized.

Scope

Our team reviewed the first version of the MPC Protocol, which still lacks certain security-relevant components. The Worldcoin team noted that a second version of the MPC Protocol is planned, which will introduce, for example, the encryption of the queries that contain shares of the iris code. We therefore recommend performing a comprehensive, follow-up security audit of the MPC Protocol once the development of version 2 is complete. Additionally, our team did not check the correctness of the specification. We further recommend performing a specification audit of the final version of the MPC protocol specification, which should include a security proof.

Dependencies

We examined all the dependencies implemented in the codebase using Rust's `cargo audit` and Go's `nancy`. We identified four issues and two warnings in the dependencies of the `mpc-uniqueness-check`. We recommend improving dependency management (Issue A).

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: Update and Replace Vulnerable Dependencies	Resolved
Suggestion 1: Include Division-By-Zero Edge Case in the Fractional Hamming Distance Computation	Resolved

Issue A: Update and Replace Vulnerable Dependencies

Location

`mpc-uniqueness-check/Cargo.toml`

`mpc-uniqueness-check/Cargo.lock`

Synopsis

We found four issues in the dependencies of the `mpc-uniqueness-check`.

Technical Details

We found the following dependency vulnerabilities in the `mpc-uniqueness-check` codebase using Rust's `cargo audit`:

- RUSTSEC-2024-0020: stack buffer overflow for dependency `whoami`;
- RUSTSEC-2023-0071: potential key recovery through timing side channels for dependency `rsa`;
- RUSTSEC-2024-0021: memory corruption for dependency `eyre`; and
- RUSTSEC-2024-0019: return of invalid tokens for named pipes (Windows-specific) for dependency `mio`.

In addition, running `cargo audit` revealed two warnings:

- RUSTSEC-2021-0141: dotnev is unmaintained and should not be used in production code; and
- iana-time-zone V0.1.59: iana-time-zone version 0.1.59 has been yanked.

Remediation

We recommend following a process that emphasizes secure dependency usage to avoid introducing vulnerabilities to the `mpc-uniqueness-check`, which includes:

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;
- Replacing unmaintained dependencies with more secure alternatives, if possible (more specifically, alternatives that are used, tested, and audited and that work as intended);
- Pinning dependencies to specific versions, including pinning build-level dependencies in the respective file to a specific version;
- Only upgrading dependencies upon careful internal review for potential backward compatibility issues and vulnerabilities; and
- Incorporating an automated dependency security check into the CI workflow, such as `cargo_audit`, `gosec`, or `nancy`.

Status

The Worldcoin team has updated the dependencies, included patched versions with fixes, and added cargo audit to the CI. The `rsa` crate affected by RUSTSEC-2023-0071 has not been upgraded since it is only used by `sqlx-mysql` within `sqlx`. Since this codebase uses `postgres`, the code is not affected. However, our team noted a new warning currently appears — RUSTSEC-2024-0320 — which states that the `yaml-rust` crate used by the `config` file is unmaintained. The Worldcoin team responded that they will upgrade it in the next release.

Verification

Resolved.

Suggestions

Suggestion 1: Include Division-By-Zero Edge Case in the Fractional Hamming Distance Computation

Location

`mpc-uniqueness-check/main/src/template.rs#L67`

Synopsis

According to the specification, the fractional hamming weight of a masked bit vector a is defined as `fhw(a) = popcount(a) / count(a)` where `count(a)` counts the number of available (unmasked) bits (set and unset), and the division operator is defined in `float`. If the edge case `count(a)=0` occurs, this will lead to a division by zero exception that is currently not handled.

Mitigation

We recommend implementing the division-by-zero edge case, or adding a check to verify that `count(a)` is always above a specified positive threshold.

Status

The Worldcoin team has added a check to handle the case where `count(a)` is zero.

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.