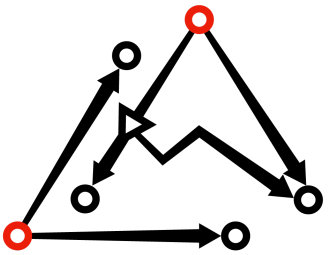

Least Authority

PRIVACY MATTERS

Cryptography Second Review

Security Audit Report

Worldcoin

Updated Final Audit Report: 14 February 2024

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - System Design
    - Code Quality
    - Documentation and Code Comments
    - Scope
  - Specific Issues & Suggestions
    - Issue A: Usage of Vulnerable Dependencies
  - Suggestions
    - Suggestion 1: Update Misleading Code Comment
    - Suggestion 2: Remove Unused Status Type
    - Suggestion 3: Update Documentation
- About Least Authority
- Our Methodology

Overview

Background

Worldcoin has requested that Least Authority perform a second security audit of their cryptography – specifically, the upgrades to semaphore-mtb and the signup sequencer. Worldcoin is building a large identity and financial network as a public utility, giving ownership to everyone. They aim to create universal access to the global economy regardless of country or background, accelerating the transition to an economic future that welcomes and benefits everyone.

Project Dates

- **September 27, 2023 - October 13, 2023:** Initial Code Review *(Completed)*
- **October 17, 2023:** Delivery of Initial Audit Report *(Completed)*
- **December 6, 2023:** Verification Review *(Completed)*
- **December 7, 2023:** Delivery of Final Audit Report *(Completed)*
- **February 14, 2024:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Mehmet Gönen, Cryptography 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 upgrades to semaphore-mtb and the signup sequencer 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:

- Worldcoin `semaphore-mtb`: https://github.com/worldcoin/semaphore-mtb
- Worldcoin `signup-sequencer`: https://github.com/worldcoin/signup-sequencer

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

- Worldcoin `semaphore-mtb`: `f25f42c62ca95f48040624291dc94ba3eb0e701a`
- Worldcoin `signup-sequencer`: `c43ce789811f29763037880ad756ddd0b071ec87`

For the verification, we examined the Git revision:

- `6dbb3a64f4498a9e87988f3955e08a02ea61369e`

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

- Worldcoin `semaphore-mtb`: https://github.com/LeastAuthority/Worldcoin_Semaphore_MTB_2nd_Review
- Worldcoin `signup-sequencer`: https://github.com/LeastAuthority/Worldcoin_Signup_Sequencer_2nd_Review

    - Additionally, the scope of this audit included the differences between the revisions: https://github.com/LeastAuthority/Worldcoin-signup-sequencer/pull/13

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:

- Website: https://worldcoin.org
- Protocol: https://whitepaper.worldcoin.org/technical-implementation#worldcoin-protocol

In addition, this audit report references the following documents:

- 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*, 2019, [GKR+19]
- K. Gurkan, K. W. Jie, and B. Whitehat, "Semaphore: Zero-Knowledge Signaling on Ethereum." *Ethereum Foundation*, 2020, [GJW20]
- Semaphore Protocol: https://semaphore.appliedzkp.org/docs/glossary#trusted-setup-files
- Semaphore Trusted Setup MPC Data: https://storage.googleapis.com/trustedsetup-a86f4.appspot.com/semaphore/semaphore_top_index.html
- Worldcoin Protocol Cryptography Security Audit Report by Least Authority.pdf, delivered on July 26, 2023
- Worldcoin Signup Sequencer Service Security Audit Report by Least Authority.pdf, delivered on August 7, 2023

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation, including cryptographic constructions and primitives;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks on the code;
- Key management, including secure key storage and proper management of encryption and signing keys;
- Exposure of any critical information during user interactions;
- Resistance to DDoS (Distributed Denial of Service) and similar attacks;
- Vulnerabilities in the code leading to adversarial actions and other attacks;
- Protection against malicious attacks and other methods of exploitation;
- Performance problems or other potential impacts on performance;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions, privilege escalation, and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team conducted a second review of the upgrades to Worldcoin's semaphore-mtb and the signup sequencer.
Previously, our team had conducted a review of an earlier version of the codebase, which included Worldcoin's Rust implementation of the `semaphore` protocol and the Go implementation of the Semaphore Merkle Tree Batcher (SMTB).

The primary areas of focus of this audit were the updated circuits for insertion and deletion, the updates in the signup sequencer, and its influence on the interaction between L1 and L2. We specifically examined the new tree and data bank structures as well as their integration with the existing codebase and could not identify any issues.

We found that the Poseidon and Keccak implementations remained consistent with previous versions, and that Poseidon's round numbers were unchanged. Similarly, our team found that the number of rounds previously reported for the Poseidon hash function is also unchanged.

Our audit of the signup sequencer was based on the same assumptions listed in our previous report, namely that only Worldcoin can make calls to the API functions `delete_identity` and `recover_identitiy`, and, hence, any attempt to delete, recover, or add a commitment is assumed to be a valid signup attempt. We also assumed `src/contracts/abi.rs` works as intended.
Our team investigated the synchronization between the database, Merkle trees, and the smart contract during concurrent modifications and could not identify any issues.

System Design

The Worldcoin system design and cryptographic implementation of the components that were in scope for this audit demonstrate strong considerations for security, which is in line with our observations as detailed in the previous security audit reports (Worldcoin Protocol Cryptography and Worldcoin Signup Sequencer).
However, our team also noted that the system can still operate in a permissioned environment only.

Code Quality

The Rust code of the updated protocol is well-organized and adheres to best practices.

Tests

The codebase includes sufficient tests, which is consistent with our previous observations.

Documentation and Code Comments

Although the documentation provides a general overview of the protocol, there are areas that could benefit from further elaboration. Specifically, the versioned tree data structure's complexity warrants a detailed description or specification.
In addition, a comprehensive flow diagram would also be beneficial for future audits and maintenance (Suggestion 3).
Additionally, while the code is generally well-commented, we recommend checking that all comments are accurate and relevant (Suggestion 1).

Scope

The scope of this review was sufficient and included all security-critical components. Note that the formal verification attempts, as specified here, were out of the scope of this audit.

Dependencies

We examined the dependencies implemented in the codebase and identified several issues. 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: Usage of Vulnerable Dependencies	Resolved
Suggestion 1: Update Misleading Code Comment	Resolved
Suggestion 2: Remove Unused Status Type	Resolved
Suggestion 3: Update Documentation	Resolved

Issue A: Usage of Vulnerable Dependencies

Location

`main/.cargo`

Synopsis

We found exploitable issues in the dependencies of the repository listed above.

Impact

High. Using unmaintained dependencies or packages with known vulnerabilities may lead to critical security vulnerabilities in the codebase.

Technical Details

- (High) RUSTSEC-2023-0065 in `tungstenite 0.17.3`: Tungstenite allows remote attackers to cause a denial of service;
- (High) RUSTSEC-2023-0052 in `webpki 0.21.4`: CPU denial of service in certificate path building;
- (Informational) RUSTSEC-2021-0139 in `ansi_term 0.12.1`: `ansi_term` is unmaintained;
- (Informational) RUSTSEC-2020-0168 in `mach 0.3.2`: mach is unmaintained;
- (Informational) RUSTSEC-2023-0040 in `users 0.11.0`: `users` crate is unmaintained;
- (Informational) RUSTSEC-2021-0145 in `atty 0.2.14`: Potential unaligned read; and
- (Informational) RUSTSEC-2023-0059 in `*const *const c_char pointer 0.11.0`: Unaligned read of `*const *const c_char` pointer.

Remediation

We recommend following a process that emphasizes secure dependency usage to avoid introducing vulnerabilities, which includes:

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;
- Replacing unmaintained dependencies with secure and battle-tested alternatives, if possible;
- 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`.

Status

The Worldcoin team has removed the `webpki` dependency and noted that `tungstenite` is an ethers dependency that is used for establishing web socket connections. However, `tungstenite` is not used in the sequencer.

Verification

Resolved.

Suggestions

Suggestion 1: Update Misleading Code Comment

Location

`main/src/app.rs#L394-L399`

Synopsis

A code comment in the specified location is misleading and may lead to confusion for developers or reviewers.

Mitigation

We recommend removing or updating the misleading comment to accurately reflect the code's intent or functionality.

Status

The Worldcoin team has updated the code comment as suggested.

Verification

Resolved.

Suggestion 2: Remove Unused Status Type

Location

`src/identity_tree.rs#L48`

Synopsis

The status type `failed` is specified but is not utilized in any functional way in the codebase. Retaining unused status types can complicate the understanding of the code's logic.

Mitigation

We suggest removing the `failed` status from the `enum` to improve clarity.

Status

The Worldcoin team has addressed this suggestion by removing the statuses `New` and `Failed` from the `enum` status.

Verification

Resolved.

Suggestion 3: Update Documentation

Location

`Readme.md#introduction`

Synopsis

The current documentation does not include details on the new API calls for deletion and recovery requests, as found in `src/server/mod.rs`. This omission could lead to an incomplete understanding or misuse of the API.

Mitigation

We recommend updating the documentation to incorporate details on the new API calls, ensuring comprehensive and up-to-date guidance for users and developers.

Status

The Worldcoin team has updated the documentation as suggested.

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.