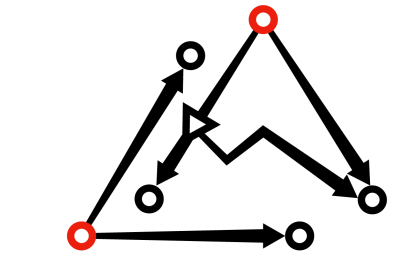

Least Authority

PRIVACY MATTERS

Pickles

Security Audit Report

Mina

FInal Audit Report: 13 December 2023

Table of Contents

Overview

Background

Mina Foundation has requested that Least Authority perform a security audit of Pickles, Mina's inductive zk-SNARK composition system.

Project Dates

- **September 18, 2023 - November 13, 2023**: Initial Code Review *(Completed)*
- **November 15, 2023**: Delivery of Initial Audit Report *(Completed)*
- **December 7, 2023:** Verification Review *(Completed)*
- **December 13, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Mehmet Gönen, Cryptography Researcher and Engineer
- Jasper Hepp, Security Researcher and Engineer
- Anna Kaplan, 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 Pickles 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:

- Pickles: https://github.com/MinaProtocol/mina/tree/develop/src/lib/pickles
- Pickles Base: https://github.com/MinaProtocol/mina/tree/develop/src/lib/pickles_base
- Pickles Types: https://github.com/MinaProtocol/mina/tree/develop/src/lib/pickles_types
- Mina Book:
    - https://github.com/o1-labs/proof-systems/tree/master/book
    - https://github.com/o1-labs/proof-systems/commit/4a87b0831c6230b572705f25eb664ca28e778bd4

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

- `1c7749f580c005d2f857755c02786c471dd87486`

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

- Mina Pickles: https://github.com/LeastAuthority/mina_pickles

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:

- Mina Docs: https://docs.minaprotocol.com
- Mina Book: https://o1-labs.github.io/proof-systems/introduction.html
- MIPS/mip-kimchi.md: https://github.com/MinaProtocol/MIPs/blob/main/MIPS/mip-kimchi.md
- 22kB-Sized Blockchain — A Technical Reference: https://minaprotocol.com/blog/22kb-sized-blockchain-a-technical-reference

In addition, during our review, our team referred to the following documents and sources:

- J. Adámek, S. Milius, and L. S. Moss, "Initial algebras and terminal coalgebras: a survey." 2010, [AMM10]
- J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Coda: Decentralized Cryptocurrency at Scale." *IACR Cryptology ePrint Archive*, 2020, [BMR+20]
- S. Bowe, J. Grigg, and D. Hopwood, "Recursive Proof Composition without a Trusted Setup." *IACR Cryptology ePrint Archive*, 2019, [BGH19]
- Q. Dao, J. Miller, O. Wright, and P. Grubbs, "Weak Fiat-Shamir Attacks on Modern Proof Systems." *IACR Cryptology ePrint Archive*, 2023, [DMW+23]
- A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge." *IACR Cryptology ePrint Archive*, 2022, [GWC22]
- R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zkSNARKs without trusted setup." *IACR Cryptology ePrint Archive*, 2017, [WTS+17]
- Pickles - Mina Book: https://o1-labs.github.io/proof-systems/specs/pickles.html
- Kimchi - Mina Book: https://o1-labs.github.io/proof-systems/specs/kimchi.html
- Halo 2 book: https://zcash.github.io/halo2
- Kimchi codebase: https://github.com/o1-labs/proof-systems
- Blog post, “A More Efficient Approach to Zero Knowledge for PLONK”: https://minaprotocol.com/blog/a-more-efficient-approach-to-zero-knowledge-for-plonk
- Blog post, “Kimchi: The latest update to Mina’s proof system”: https://minaprotocol.com/blog/kimchi-the-latest-update-to-minas-proof-system
- Blog post, “Meet Pickles SNARK: Enabling Smart Contracts on Mina Protocol”: https://medium.com/minaprotocol/meet-pickles-snark-enabling-smart-contract-on-coda-protocol-7ede3b54c250
- ZK-GLOBAL 0x05 - Izaak Meckler - Meet Pickles SNARK: https://www.youtube.com/watch?v=nOnGOxyh7jY
- The Pickles Inductive SNARK Composition System: https://www.youtube.com/watch?v=ZQkzTB8VDzs

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the Pickles implementation;
- Correctness of the strong Fiat Shamir heuristic;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution – excluding attacks that require controlling the circuits;
- 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

Mina implements a succinct blockchain that uses Kimchi and Pickles for transparent, recursive zero-knowledge proofs based on Halo 2 [BGH19] and Plonk [GWC22] in order to maintain a small blockchain size, ensuring efficiency and privacy while supporting decentralized applications and smart contracts. The Kimchi proof system is written in Rust, while Pickles is written in OCaml.

System Design

Our team found that security has been strongly considered in the design and implementation of Pickles as demonstrated, for example, by the usage of OCaml's type system.

In our review, we compared the code against the Plonk and Halo 2 specifications and did not identify any security-relevant issues. We examined the step and wrap prover as well as their verifiers in Pickles to investigate whether they are implemented correctly and could not find any issues.

Kimchi and Pickles implement complex, state-of-the-art protocols with security heavily based on a properly implemented multi-move Fiat Shamir construction. We reviewed the correctness of the strong Fiat Shamir implementation of a multi-move interactive oracle proof (IOP) (explained in [DMW+23]) by examining the sequential Fiat Shamir steps, both in Plonk as well as Mina's adaptation of Halo 2. Our team investigated whether a sufficient transcript of all appearing statements, as well as all public knowledge that the simulated verifier has at each step in the sequential proof, are included into the transcript and properly hashed into simulated challenges. We did not find any issues in Kimchi and in Pickles, nor in Mina's adoption of Halo 2, nor in the recursive verifiers.

We reviewed Mina's concept of inductive rules, in addition to investigating how OCaml's type system ensures that custom rules indeed define an inductive system in the abstract sense (as discussed in [AMM10]) and could not identify any issues.

We also analyzed the implementation of – and thought process behind – deferred values since deferring computation for certain values is needed, as a two-cycle of elliptic curves is necessary for Mina's inductive proof system.

Code Quality

We performed a manual review of the repositories in scope and found the codebases to be generally well-organized, well-written, and of high quality. However, our team found that variables are used inconsistently across the codebase and that resolved TODOs have not been removed from the codebase (Suggestion 2).

Documentation and Code Comments

The project documentation provided for this review offers a generally sufficient overview of the system and its intended behavior. However, our team found that the Mina book had multiple typographical errors and deviated from the code and reference papers in certain places. A lack of clear and comprehensive documentation hinders the ability to understand the intention of the code, which is critical for assessing the security and correctness of the implementation. We recommend improving the documentation (Suggestion 1).

During the review, the Mina team was readily available and addressed our team's questions and concerns by updating the documentation (e.g. in pull requests here and here) and adding comments to the Pickles codebases (e.g. in an open pull request here).

Our team noted that code comments have been improved since our previous audit of the Mina Foundation's Transaction Logic and Pool, which our team completed and delivered on August 28, 2023. We found that code comments sufficiently describe the intended behavior of security-critical components and functions.

Scope

While the Kimchi proof system was out of scope for this audit, our team had to partially review it in order to build a better understanding of proof creation and the Fiat Shamir heuristic to effectively audit Pickles. Due to the scope definition, our team was unable to analyze the implementation and security of the lookup functionality as well as the custom gates.

We recommend that Kimchi, and in particular the correct implementation and security of the lookup functionality and the custom gates, be comprehensively audited by an independent security company familiar with the Mina codebase to improve the security of the system.

Dependencies

Our team found that outdated libraries are used, as we noted in Suggestion 1 in our previous report on Mina's Transaction Logic and Pool. We recommend that well-audited and maintained dependencies be used.

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
Suggestion 1: Improve Documentation	Partially Resolved
Suggestion 2: Improve Variable Name Consistency and Update TODOs	Planned

Suggestions

Suggestion 1: Improve Documentation

Location

`Mina book`

`src/evaluation_proof.rs#L331-L335`

Synopsis

Mina is a highly complex system that implements state-of-the art cryptography and provides crypto assets for its users. Although the code is written in OCaml – a programming language characterized by its functional programming features and high security in terms of type system features – the complexity of the language reduces the readability of the code and, thus, makes reasoning about the security of the system more difficult.
As a result, it is highly important that both future developers and auditors have clear and precise specifications, in addition to detailed documentation, to easily understand the different components of the system.
For this purpose, the Mina team created the Mina book, which is currently a work in progress.

However, during the audit, our team discovered multiple typographical errors, imprecisions, deviations from the code, and missing documentation. For example, we identified:

- Outdated sections on snarky-rs in the Mina book (the team has removed this already in this PR);
- Typographical errors in the Mina book (the team has already fixed them in this PR);
- Deviation from the definition in [BGH19] of `r'` in the Kimchi code (due to the respective changes of `a^j`, `b^j`, and `G^j`, as described here in the Halo 2 book);
- Missing `meOnly` in the documentation (see here); and
- Missing documentation on critical cryptographic roles (e.g., the use of hash functions, such as `md5`).

Therefore, in its current state, the book cannot fully serve as a reliable specification for future developers implementing new features and fixing bugs, as well as auditors conducting a comprehensive security audit of Mina.

Mitigation

We recommend addressing the aforementioned suggestions to improve the documentation.

Status

The Mina team stated that the documentation was out of scope for this audit. However, our team noted that auditors often refer to the project documentation during the review to build a comprehensive understanding of the system and, hence, it cannot be entirely excluded from the scope considerations. As such, we recommend that the Mina team continue to look for opportunities to improve the documentation to allow for better maintenance and future audits.

Verification

Partially Resolved.

Suggestion 2: Improve Variable Name Consistency and Update TODOs

Location

Examples (non-exhaustive):

`lib/pickles/wrap.ml#L126-L127`

`pickles/plonk_checks/plonk_checks.ml#L309`

Synopsis

During our review, we found that variables often have different names in the specification, the underlying research papers, the Kimchi proof system code, and the Pickles code. This significantly increases the effort needed to understand potentially security-critical components of the system and hence makes reasoning about the security of the system more difficult. For example, in the code: `v`, `z1`, `polyscale`, and `xi` all refer to the same variable across Pickles and Kimchi.

In addition, we found unresolved TODOs in the codebase. Addressing TODOs prior to a comprehensive security audit of the code allows security researchers to better understand the full intended functionality of the code, indicates completion, and increases readability and comprehension.

Mitigation

We recommend updating the names of the variables, such that they have accurate, descriptive, and consistent names and removing the TODOs.

Status

The Mina team acknowledged that the naming of variables and comments can and should be improved in Pickles and Kimchi but noted that they can only start planning the renaming of `code-churn` after their Berkeley release is finished.

Verification

Planned.

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.