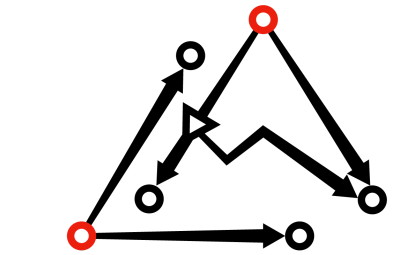

Least Authority

PRIVACY MATTERS

Transaction Logic & Transaction Pool

Security Audit Report

Mina Foundation

Updated Final Audit Report: 28 August 2023

Table of Contents

Overview

Background

Mina Foundation has requested that Least Authority perform a security audit of their Transaction Logic and Transaction Pool.

Project Dates

Audit 1: Transaction Logic

- **March 30, 2023 - May 24, 2023:** Initial Code Review *(Completed)*
- **May 26, 2023**: Delivery of Initial Audit Report *(Completed)*
- **August 14, 2023:** Verification Review *(Completed)*
- **August 16, 2023:** Delivery of Final Audit Report *(Completed)*
- **August 28, 2023:** Delivery of Updated Final Audit Report *(Completed)*

Audit 2: Transaction Pool

- **April 25, 2023 - May 25, 2023**: Code Review *(Completed)*
- **May 29, 2023**: Delivery of Initial Audit Report *(Completed on May 26, 2023)*
- **August 14, 2023:** Verification Review *(Completed)*
- **August 16, 2023:** Delivery of Final Audit Report *(Completed)*
- **August 28, 2023:** Delivery of Updated 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 the Transaction Logic and Transaction Pool implementation of the Mina Protocol written in OCaml followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following folders, files, and code repositories are considered in scope for the review:

Audit 1: Transaction Logic

- Mina Transaction Logic: https://github.com/MinaProtocol/mina/blob/develop/src/lib/transaction_logic/mina_transaction_logic.ml
- ZkApp Command Logic: https://github.com/MinaProtocol/mina/blob/develop/src/lib/transaction_logic/zkapp_command_logic.ml
- Transaction SNARK: https://github.com/MinaProtocol/mina/blob/develop/src/lib/transaction_snark/transaction_snark.ml
- Snarked Ledger State: https://github.com/MinaProtocol/mina/blob/develop/src/lib/mina_state/snarked_ledger_state.ml

Audit 2: Transaction Pool

- Transaction Pool: https://github.com/MinaProtocol/mina/blob/develop/src/lib/network_pool/transaction_pool.ml
- Prod: https://github.com/MinaProtocol/mina/blob/develop/src/lib/verifier/prod.ml
- Common: https://github.com/MinaProtocol/mina/blob/develop/src/lib/verifier/common.ml

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

- `dfc3a21ce43904bd9d23a3d3ce2c3a2c570071f2`

For the verification, we examined the following issues and pull requests :

- https://github.com/MinaProtocol/mina/issues/13252
- https://github.com/MinaProtocol/mina/pull/13048
- https://github.com/MinaProtocol/mina/issues/13253
- https://github.com/MinaProtocol/mina/issues/13277

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

- Mina Protocol: https://github.com/LeastAuthority/Mina_Protocol

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 Technical Whitepaper (Coda: Decentralized Cryptocurrency at Scale): https://eprint.iacr.org/2020/352.pdf
- Mina Docs: https://docs.minaprotocol.com
- Mina's Coding Style Recommendations: https://docs.minaprotocol.com/node-developers/style-guide

- Mina book: https://o1-labs.github.io/proof-systems/introduction.html
- Mina: Economics and Monetary Policy: https://minaprotocol.com/wp-content/uploads/economicsWhitepaper.pdf
- [MIPS]: https://github.com/MinaProtocol/MIPs/tree/main
- Base58Check: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/base58_check/README.md
- Child processes: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/child_processes/README.md
- Crypto: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/crypto/README.md
- Kimchi Backend: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/crypto/kimchi_backend/README.md
- Kimchi Bindings: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/crypto/kimchi_bindings/README.md
- Kimchi: https://github.com/o1-labs/proof-systems/blob/373a294cf87f94d1bb518a248fbf71364aa73526/README.md
- Snarky Tests: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/crypto/snarky_tests/README.md
- Merkle ledgers: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/merkle_ledger/README.md
- Notes on super catchup run logic: https://github.com/LeastAuthority/Mina_Protocol/blob/develop/src/lib/ledger_catchup/README.md
- What is Ouroboros Samisika? https://minaprotocol.com/blog/what-is-ouroboros-samisika

In addition, this audit report references the following documents and links:

- A. Bariant, C. Bouvier, G. Leurent, and L. Perrin, "Algebraic Attacks against Some Arithmetization-Oriented Primitives." *IACR Transactions on Symmetric Cryptology*, 2022, [BBL+22]
- 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]
- L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A Faster Version of the Poseidon Hash Function." *IACR Cryptology ePrint Archive*, 2023, [GKS23]
- M. Schofnegger, "calc_round_numbers.py." 2021, [S21]
- E. Schorn and A. Howell, "Mina Client SDK, Signature Library and Base Components – Cryptography and Implementation Review." *NCC Group,* 2022, [SH22]
- How to Comment Programs: https://ocaml.org/docs/guidelines#how-to-comment-programs
- OCaml Documentation Guidelines: http://ocamlverse.net/content/documentation_guidelines.html

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the logic;
- Attacks that impacts funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of service (DoS) and other security exploits that would impact the intended use of the logic or disrupt its execution;
- Vulnerabilities in the code;
- Proper management of encryption;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Mina is a sophisticated succinct blockchain that utilizes zk-SNARKs and the Ouroboros Samisika Consensus (see technical whitepaper and blog). The Mina blockchain achieves succinctness by implementing both the Transaction Logic as well as the Consensus Logic as a recursive zk-SNARK. This novel approach introduces complex circuits and logic. Mina's zk-SNARKs are based on advanced cryptographic protocols like PLONK with custom gates and are implemented in the OCaml language, which enables developers to write code that meets the safety requirements of high security software.

Mina intends to introduce zero-knowledge smart contracts called ZkApps in a hard fork, including privacy and off-chain computation features (see MIP). ZkApps are a system for updateable off-chain smart contracts, whose internal logic is invisible on-chain. This enables scalability and allows for new use cases and applications previously considered impossible with traditional smart contracts. The chain only holds an updateable verification key, and a small on-chain state to verify a zk-SNARK that proves proper smart contract execution.

The introduced changes affect the processing of transactions especially. Our team audited the Transaction Pool, where transactions are added to the mempool, and the Transaction Logic, where transactions are added to a block and verified by the Transaction SNARK.

The introduced changes shift much of the functionality that is traditionally performed on layer 2 into layer 1 of the blockchain, substantially increasing protocol complexity. In this case, Mina layer 1 would support custom tokens as well as built-in vesting periods and token lock-ups. Moreover, an elaborate system of permissions and preconditions is implemented, where it is difficult to reason about all the possible combinations.

Our team performed a comprehensive review of the code in scope and identified issues and areas of improvement in the design and implementation of the Transaction Logic and the Transaction Pool, as detailed below and outlined in the Issues and Suggestions of this report.

Implementation

Our team found that the codebase is well-organized, and the implementation generally adheres to Mina's coding style conventions.

During our review, we compared the implementation of the ZkApp system to the MIP describing them and identified a diversion between the implementation and the MIP (Suggestion 2). In addition to the areas of concern listed above, our team investigated several areas for security vulnerabilities, including:

- Potential overflow and underflow issues when adding balances and updating nonces;
- Transaction replay attacks for ZkApps based on the MIP;
- The logic of the verification key within the Transaction Logic and Transaction Pool, and the key's hash table in the Transaction Pool;
- The logic in the Transaction Pool dealing with ZkApp commands in the functions `apply`, `verify`, `create`, and `handle_transition_frontier`;
- The logic of the account creation fee;
- Possible denial-of-service (DoS) / distributed denial-of-service (DDoS) attack vectors against the Transaction Pool;
- Gaming attacks against Mina's protocol;
- Issues in non-trivial call forests; and
- Missing signature and proof verification checks.

In our review, our team found missing checks in the Transaction Logic and SNARK that can lead to severe vulnerabilities (Issue D). We also found redundant checks in the Transaction SNARK that reduce the efficiency of the system (Issue E) as well as missing checks during the (re)validation of transactions in the pool (Issue A, Suggestion 6).

Furthermore, our team noted that the implementation uses the Poseidon hash function. However, the configuration of the hash function results in a security level that is currently less than the claimed 128-bits but still within the recommended threshold set by NIST. We recommend documenting the security level correctly (Suggestion 5).

We also noted that there is an open PR listing five issues relating to missing permission checks in the Transaction Pool, out of which we found one.

Tests

Our team found that sufficient tests are implemented.

Documentation & Code Comments

The documentation that was available for this review was insufficiently organized and, in some cases, outdated. As a result, our review team was not able to efficiently learn the details of the system. For example, it was challenging for the review team to understand how preconditions or action states are intended to work. We recommend that the missing documentation be added to facilitate current and future analysis of the system (Issue B).

Additionally, there is no technical specification documenting a detailed circuit design for the zk-SNARKs. Our team noted that parallel efforts to write a complete circuit specification, while writing code, would greatly improve the readability of the code by documenting pre and post conditions (Issue C).

In addition, we found that there were insufficient code comments describing the intended behavior of code that is critical to the security of the implementation (Suggestion 4). Reasoning about the functionality of code written in OCaml without code comments results in inefficiencies, which increases the possibility of vulnerabilities being overlooked.

Scope

The scope of this review included all relevant components. Although our team was able to review all the in-scope code, a thorough analysis of the system was hindered due to the partial documentation and the

limited availability of the Mina development team. Subsequently, our review was not as comprehensive as usually preferred for the complexity of such a system. We encourage the Mina development team to provide additional resources for future reviews to increase the efficiency of the audits and ultimately decrease the attack surface.

For example, our team was unable to learn about the intended functionality of `Checked.t` and `run_checked`. These functions are an integral part to understanding the zk-SNARK's constraint system.

Additionally, Mina implements large parts of its logic in a circuit that compiles into a zero-knowledge proving system called Kimchi. According to the MIP of the Kimchi proving system, edge cases can occur in writing Kimchi circuits that developers and auditors must be aware of. Since there are open questions about those edge cases and the general functionality of the Kimchi prover system, we were unable to comprehensively reason about a class of attacks that results from compromised soundness due to missing constraints. We were also unable to analyze the `call stack` and the `stack frame` in detail.

Furthermore, although Kimchi and snarky.js are fundamental building blocks in the implementation, and are most critical for system security, they have not been audited by external third parties. We recommend a series of audits starting with the core cryptography components and then working up through the entire Mina codebase. While some older reports are listed on the Mina Docs website, they are out of date. Hence, our team was unable to find any reports on security reviews of the technology underlying the cryptographic stack to be used by Mina.

We recommend that the Mina team work closely with external third parties to bridge the gaps in security audits of core components, and bring the protocol documentation up to working standards, thereby increasing security and reducing obfuscation.

Dependencies

Our team reviewed the use of dependencies in the implementation of Mina's Transaction Logic and Transaction Pool and found that outdated libraries are used. We recommend that up-to-date and well audited and maintained dependencies be used (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 Update in Transaction Pool on Verification Key	Unresolved
Issue B: Missing Documentation	Unresolved
Issue C: Missing Protocol Specification	Unresolved
Issue D: Signed Command Logic Is Not Checking Signature by Source Account	Resolved
Issue E: Redundant Checks In SNARK Decrease Efficiency	Unresolved
Issue F: Delegate for Staking Can Be Used for Non-Default Tokens	Resolved
Suggestion 1: Update Outdated Dependencies and Build Warnings	Unresolved

Suggestion 2: Document Deviations from the MIPs	Unresolved
Suggestion 3: Use the Total Fee of Sender Queues for Pruning	Unresolved
Suggestion 4: Increase Code Comments	Unresolved
Suggestion 5: Increase Poseidon's Total Number of Rounds or the Degree of S-Boxes	Partially resolved
Suggestion 6: Add Expiration Check When Handling New Transition Frontiers in the Transaction Pool	Unresolved

Issue A: Missing Update in Transaction Pool on Verification Key

Location

`src/lib/network_pool/transaction_pool.ml#L768`

Synopsis

During revalidation of the Transaction Pool against the new best tip ledger, verification keys of dropped commands are not deleted from the verification key hash table. This can lead to a memory leak.

Impact

Block producers collect valid transactions in the Transaction Pool. The Consensus Protocol can lead to a change in the current chain, and the best tip ledger is updated. In this case, the transactions in the pool need to be revalidated based on the new best tip ledger. If commands are dropped, any related data needs to be deleted from the pool. Here, the function `create` fails to delete the associated verification key from the hash table. Although this could lead to a memory leak, we did not identify any particular attack vector, and the overall impact is low.

Feasibility

The memory leak would occur over time if the protocol switches the chain and needs to drop commands based on this.

Remediation

We recommend deleting the data from the hash table when commands are dropped.

Status

The Mina team created a PR to address this Issue. The team also stated that the attack is very difficult to execute, and the effect of the attack is a small one-time data leak on each node targeted, as the leak only occurs if the node had previously been tricked onto a malicious long fork during bootstrap, and then later discovers the honest chain. Hence, at the time of the verification, the suggested remediation has not been resolved.

Verification

Unresolved.

Issue B: Missing Documentation

Location

`Mina Protocol`

`MinaProtocol/mina`

Synopsis

Mina's codebase lacks proper documentation that explains the design, data flow, and structure of the system, as well as the data structures used in the code. The following cases serve as examples:

- The `voting_for` field for account updates in the MIP is not well-explained. It is not clear how it is linked to ZkApp commands or on-chain voting;
- The verification key is not documented well enough for ZkApps developers. In addition, the verification key hash table as well as side loading are not documented well enough to reason about all attack vectors when they are being utilized for ZkApps;
- The `action_state` update is not well-documented, and the reason for updating it only once per global slot number is not well-documented;
- It is unclear if it is possible to use the `delegate` field for a non-default token ID in ZkApps;
- Different states are not properly described in the context of Mina. For example: `ledger`, `stackframe`, `call_forest`, `local_state`, `global_state`, `previous_state`, `call_stack`, `slot`, `epoch`, `best_tip_ledger`, `transaction_frontier`; and
- The documentation on canceling transactions is not up to date on the website. It is currently possible to cancel a transaction, although the documentation states otherwise.

Impact

Without proper documentation on the behavior, logic, and intended use cases, the system's security partly relies on obfuscation. As a result, it is difficult to reason about edge cases that might lead to security issues.

Remediation

We recommend that the Mina team gather previous and current Github discussions in parallel with code development, and create general documentation. This task could be outsourced to a qualified team with close knowledge of the Mina ecosystem.

Status

This Issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue C: Missing Protocol Specification

Location

`Mina Protocol`

`MinaProtocol/mina`

Synopsis

No formal specification of Mina's protocol exists that can serve as an independent source of truth, and the system is currently only realized as an insufficiently commented and documented OCaml

implementation. The specification as well as the reasoning for the design choices are scattered across hundreds of Github issues and pull requests (e.g. here).

Furthermore, no formal specification of Mina's transaction SNARK exists, and the whitepaper is outdated.

Impact

Without a protocol description independent of any particular implementation, the system's security partly relies on obfuscation, and reasoning about issues resulting from missing domain separation of signatures, proofs, and the other existing cryptographic building blocks is very difficult to near impossible. Without a proper formal description of the transaction SNARK's underlying circuit, reasoning about the soundness of the SNARK is extremely difficult.

Remediation

We recommend that the Mina team work in parallel with code development and write a proper protocol specification, including a statement description for Mina's zk-SNARKs. Such a description is necessary for third parties to implement Mina nodes in different languages and reason about protocol security. This task could be outsourced to a qualified team with close knowledge of the Mina ecosystem.

Status

This Issue remains unresolved at the time of verification.

Verification

Unresolved.

Issue D: Signed Command Logic Is Not Checking Signature by Source Account

Location

`src/lib/transaction_logic/mina_transaction_logic.ml`

`src/lib/transaction_logic/mina_transaction_logic.ml#L830-L838`

Synopsis

A missing check in the Transaction Logic allows the transfer of funds from any account.

A signed command is a type of transaction that transfers funds from a source account to a receiver account with a fee payer account to pay the fee. For a signed command, the fee payer must be equal to the source account.

A check to enforce equality between the public keys of the fee payer and the source account is missing in the Transaction Logic and the Transaction SNARK. Since it is also not checked that the source account signed the transaction, it is possible to forge a payment that is not signed by the source, but by some arbitrary account, as long as that account pays the fee.

Impact

Critical. The missing check opens the possibility to transfer funds from arbitrary accounts.

Feasibility

The attack is straightforward. Currently, the GraphQL API prevents the creation of such a transaction but it is trivial to change it.

Remediation

We recommend that the Mina team add checks to verify that the source account signed the transaction.

Status

The Mina team has implemented the remediation as recommended.

Verification

Resolved.

Issue E: Redundant Checks In SNARK Decrease Efficiency

Location

src/lib/transaction_logic/zkapp_command_logic.ml#L1163

src/lib/transaction_logic/zkapp_command_logic.ml#L1208

src/lib/transaction_logic/zkapp_command_logic.ml#L1739-L1743

src/lib/transaction_logic/zkapp_command_logic.ml#L1818-L1842

Synopsis

We identified several redundant operations in the ZkApp command logic. These increase constraints in the SNARK and decrease efficiency. We found the following instances of redundant code:

- The account inclusion proof is performed twice, once in L1163 and again in L1208;
- `local_state.token_id` is always set to `default` and `local_state.token_id` is not used, which renders any check on it redundant; and
- `local_state.excess` is asserted to be zero before the `global_state.excess` calculation at the end of the transaction (after the last account update). As a result, the `global_state.excess` calculation (adding the excess of the `local_state` to the excess of the `global_state`) for the last account update is redundant.

Impact

This is not a security Issue but an efficiency Issue. Redundant code in the SNARK adds unnecessary constraints, which decrease the efficiency of the protocol.

Remediation

We recommend removing the reported instances of redundant code.

Status

The Mina team created a PR to log this Issue. The team also stated that although this Issue might result in some unnecessary constraints implemented in the SNARK circuit, there is no security impact.
At the time of the verification, the suggested remediation has not been resolved.

Verification

Unresolved.

Issue F: Delegate for Staking Can Be Used for Non-Default Tokens

Location

src/lib/transaction_logic/zkapp_command_logic.ml#L1620

src/lib/transaction_logic/zkapp_command_logic.ml#L1639

Synopsis

When creating a new account with a non-default `token_id`, the field `delegate` should be None. This is done by an honest block producer when creating an account, but is not enforced in the Transaction Logic and Transaction SNARK. Consequently, a malicious block producer could set the delegate to an account with non-default tokens.

Impact

Potentially Critical. In the worst case, this would allow a malicious actor to stake with non-default tokens in the Mina protocol. By creating a custom token with a maximum supply, the actor can use this custom token in the staking and increase the weight of its signature close to one. Hence, the attacker would control the Consensus Protocol and could easily take over the block production.

However, the code that makes the attack possible in this Issue is out of scope. Additionally, at the time of writing of this report, our question to the Mina team regarding this Issue is still pending. As a result, we are uncertain about the real impact of this Issue.

Remediation

We recommend that accounts with non-default `token_id` be prevented from setting the delegate to a value other than None in the Transaction Logic and SNARK.

Status

The Mina team stated that default values are enforced within the SNARK correctly, and the blockchain SNARK prevents non-default token accounts from being used for staking. Therefore, this is not a valid Issue.

Verification

Resolved.

Suggestions

Suggestion 1: Update Outdated Dependencies and Build Warnings

Location

Examples (non-exhaustive):

opam.export

develop/README-dev.md

src/lib/crypto/kimchi_bindings/js/node_js/build.sh#L6

src/libp2p_ipc/go.mod#L3

Synopsis

Some of the dependencies used by the project are outdated:

- When running `opam switch import src/opam.export`, the warning for `conf-openssl.1` persists. This has been noted already in [SH22];
- The Rust toolchain is bound to version `nightly-2022-09-12`, while the latest version is `1.71.0`; and
- Golang is bound to version `1.18.10`, while the latest version is `1.20.4`.

Additionally, the build process is complex and yields a considerable number of warnings, which might, consequently, impact the underlying trust assumptions the system is based on.

Mitigation

We recommend updating the dependencies and reviewing the build process to make it build in a straightforward way on a freshly installed operating system of choice (ideally, several different freshly installed operating systems).

Status

The Mina team stated that they are considering the suggestion but are uncertain if this is necessary before the upcoming ZkApps hard fork release.
Since the Go and Rust libraries have not been in scope for this audit, our team cannot assess the security risk of using deprecated Go and Rust versions. Therefore, we highly recommend upgrading to the latest version due to several security fixes listed here for Go, or here for Rust (for example, changes to the elliptic curve library of Go).

Verification

Unresolved.

Suggestion 2: Document Deviations from the MIPs

Location

`MIPs - ZkApp Transactions`

Synopsis

Our team identified one implementation deviation from the MIPs. Since the document was not in scope, our team cannot verify its correctness. Moreover, it is possible that more deviations exist.

In particular, we found that the code checks preconditions against the `local_state`, which is the current state in MIP terminology. According to the MIP, these checks are performed against the previous state (the `global_state`).

As stated here, an MIP "*should provide a concise technical specification of the feature and a rationale for its inclusion in the protocol.*" Therefore, it should be possible to see which parts of an MIP are integrated into a system update, left out, or changed – and why. Since the community should discuss it and reason about it in a transparent process, any deviations need to be documented properly.

Mitigation

We recommend documenting this, and any other deviations from MIPs, properly to make it noticeable for anyone reading the MIP.

Status

The development team acknowledged the finding and stated that they shared this mitigation with the Mina Foundation to figure out the right process for updating the document. However, at the time of the verification, the suggested mitigation has not been resolved.

Verification

Unresolved.

Suggestion 3: Use the Total Fee of Sender Queues for Pruning

Location

`src/lib/network_pool/indexed_pool.ml#L621`

Synopsis

The Mina protocol frequently prunes transactions from the Transaction Pool to keep its memory footprint below a certain boundary. This is done by checking for fees across all transactions and then removing transactions with the lowest fees. To keep the pool consistent, other transactions from the same sender (ones that might not have low fees), are also deleted due to potential nonce inconsistencies. Depending on the fee distribution of transactions in the queue of a single user, this is not the most optimal strategy to prune transactions from the pool, as it might lead to situations where high fee transactions are pruned. Competitive node runners would therefore be incentivised to rewrite the code in a potentially disorganized way, possibly introducing issues to their block production.

Mitigation

We recommend partitioning the pool into queues and computing fees of entire queues. The pool should then be ordered and pruned by those fees.

Status

The Mina team acknowledged the finding but stated that they decided not to implement the mitigation. They added that due to the design of the transaction pool, they have no guarantee that the subsequent transaction in the queue will actually be applicable. Due to this, users could just ensure their low-fee transactions do not get pruned by enqueueing a series of high-fee transactions after which they will be invalidated once the first transaction is applied.

Verification

Unresolved.

Suggestion 4: Increase Code Comments

Synopsis

The codebase contains very few code comments, with a proportion of those included being outdated. Code comments increase maintainability and reduce the risk of introducing vulnerabilities later on. Specifically, SNARK logic and cryptographic operations should be explained in code comments.

Mitigation

We recommend that comprehensive code comments be implemented to describe input parameters, control flow conditions, expected return values and, if possible, pre and post conditions. Code comments should also describe the intended use of functors, modules, functions, etc. The OCaml blog provides a useful reference on how to best utilize code comments (see here and here).

Status

This suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 5: Increase Poseidon's Total Number of Rounds or the Degree of S-Boxes

Location

`Mina Book`

`src/lib/crypto/kimchi_bindings/stubs/src/lib.rs`

Synopsis

In the Mina protocol, the Poseidon hash function is implemented with 55 rounds, and the degree of S-Boxes is 7. According to best practices and guidelines, as noted in [S21], this number of rounds for Poseidon is sufficient for 128-bit security. However, it is possible to reduce this security level by applying an interpolation attack, which is defined in [BBL+22]. In section 4.3 of this research paper, the authors conduct a complexity analysis for this attack by assuming `r` is the total number of rounds for Poseidon, and `t` is the degree of S-Boxes.

In this case, the security level for the Poseidon hash function is equal to log(d*log(d)*(log(d) + log(p))*log(log(d))) where `d=t^(r-2)`, which is the degree of the univariate polynomial, and `p` is the field size. In the Mina protocol, `t=7` and `r=55`. Hence, the security level is approximately equal to 116-bit.

The attack detailed in this research would reduce the expected security of the Poseidon hash function. While this security level is slightly higher than the threshold of 112-bit security recommended by NIST, it is still below the 128-bit security generally recommended according to best practices. In addition, `Poseidon2` – a new version of the Poseidon hash function – has been designed, as explained in [GKS23], in order to mitigate the attack described above.

Mitigation

We recommend documenting the current security level of 116-bit security for the current configuration of the `Poseidon` hash function.

Status

The Mina team acknowledged the attack and the resulting lower security level. The team added that it does not seem necessary to document the reduced security level incurred by interpolation attacks, as they are already following best practices for 128-bit security, which takes such an attack into consideration.

Verification

Partially resolved.

Suggestion 6: Add Expiration Check When Handling New Transition Frontiers in the Transaction Pool

Location

src/lib/network_pool/transaction_pool.ml#L768

Synopsis

When handling new transition frontiers in the function `create`, the code does not check the expiration of transactions in the pool with the function `remove_expired`. Hence, expired transactions persist in the Transaction Pool. This is not a security Issue since the pool is updated again upon reception of a transition frontier difference in the function `handle_transition_frontier_diff`. However, it poses the risk of inefficiencies for block producers.

Mitigation

We recommend adding a check to determine whether transactions in the pool expired.

Status

The Mina team created a PR to log this suggestion. However, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, 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.