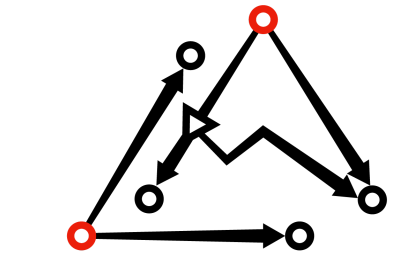

Least Authority

PRIVACY MATTERS

Plonky3

Security Audit Report

Polygon

Updated Final Audit Report: 12 July 2024

Table of Contents

Overview

Background

Polygon has requested that Least Authority perform a security audit of Plonky3. Plonky3 is a toolkit for implementing polynomial IOPs (PIOPs), such as PLONK and STARKs.

Project Dates

- **March 25, 2024 - May 9, 2024:** Initial Code Review *(Completed)*
- **May 14, 2024:** Delivery of Initial Audit Report *(Completed)*
- **June 21, 2024:** Verification Review *(Completed)*
- **June 25, 2024:** Delivery of Final Audit Report *(Completed)*
- **July 12, 2024:** Delivery of Updated 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 Plonky3 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:

- Plonky3:
  https://github.com/Plonky3/Plonky3

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

- `2edbd19b0482decf20443e90c78f375672da8241`

For the verification, we examined the Git revision:

- `7bb6db50594e159010f11c97d110aa3ee121069b`

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

- Plonky3:
  https://github.com/LeastAuthority/Polygon-Zero-Plonky3

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://polygon.technology

In addition, this audit report references the following documents:

- JP. Aumasson, D. Khovratovich, B. Mennink, and P. Quine, “SAFE: Sponge API for Field Elements.” *IACR Cryptology ePrint Archive*, 2023, [AKM+23]
- E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Scalable, transparent, and post-quantum secure computational integrity.” *IACR Cryptology ePrint Archive*, 2018, [BBH+18]
- D. Bernhard, O. Pereira, and B. Warinschi, “How not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios.” *IACR Cryptology ePrint Archive*, 2016, [BPW16]
- A. R. Block, A. Garreta, J. Katz, J. Thaler, P. R. Tiwari, and M. Zając, “Fiat-Shamir Security of FRI and Related SNARKs.” *IACR Cryptology ePrint Archive*, 2023, [BGK+23]
- K. J. Bowers, R. A. Lippert, R. O. Dror, and D. E. Shaw, "Improved Twiddle Access for Fast Fourier Transforms." *IEEE Transactions on Signal Processing*, 2010, [BLD+10]
- 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. Faonio, D. Fiore, M. Kohlweiss, L. Russo, and M. Zajac, “From Polynomial IOP and Commitments to Non-malleable zkSNARKs.” *IACR Cryptology ePrint Archive*, 2023, [FFK+23]
- L. Grassi, D. Khovratovich, and M. Schofnegger, “Poseidon2: A Faster Version of the Poseidon Hash Function.” *IACR Cryptology ePrint Archive*, 2023, [GKS23]
- U. Haböck, “A summary on the FRI low degree test.” *IACR Cryptology ePrint Archive*, 2022, [Haböck22]
- U. Haböck, D. Levit, and S. Papini, “Circle STARKs.” *IACR Cryptology ePrint Archive*, 2024, [HLP24]
- A. Green and D. Vagner, “The Multi-FRI Protocol.” (unpublished draft version)
- Blog post, “The second preimage attack for Merkle Trees in Solidity”:
  https://www.rareskills.io/post/merkle-tree-second-preimage-attack
- Field Merkle Tree in Plonky3:
  https://hackmd.io/@0xKanekiKen/H1ww-qWKa
- Barycentric interpolation:
  https://hackmd.io/@vbuterin/barycentric_evaluation

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Vulnerabilities in the code leading to adversarial actions and other attacks;
- Protection against malicious attacks and other methods of exploitation; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a comprehensive audit of Polygon’s Plonky3, a toolkit for an efficient implementation of a non-hiding STARK protocol. The codebase consists of different cryptographic building blocks (such as fields, hash functions, polynomial commitment schemes, and others) that can be combined into STARKs. The overall goal is to build a more performance efficient toolkit to improve upon existing similar codebases, such as Plonky2. Notable novel pieces are the `Mersenne31` prime field with Circle Stark

([HLP24]), a new Fast Reed-Solomon interactive oracle proof of Proximity (FRI) flavor called Multi-FRI (as explained in the unpublished draft by A. Green and D. Vagner), Mixed Matrix Merkle Tree (MMMT) Commitments (see here for a description), and complex optimizations for the Fast Fourier Transformation (FFT) (i.e., [BLD+10]).

System Design

Our team reviewed the system's design and implementation and found that security has been taken into consideration as demonstrated by the use of Rust and advanced cryptography as well as the absence of patterns in the issues identified.

During our investigation, we performed a manual code review of the components in scope and considered various attack vectors. Our team found several issues and suggestions, as outlined below.

Fields

We reviewed the implementation of the field trait in the folder `\field` as well as the fields `BabyBear`, `Mersenne_31`, `Goldilocks`, and the curve `BN254-fr`. Since the code allows for non-reduced representations of finite field elements, we investigated the implementation, in particular, for any issues that might arise from the inherent ambiguity. Apart from one missing check that our team identified (Issue D), we did not find any issues.

We noticed that the trait `Field` implements Rust's `Ord` trait, which is derived from the total order on integers. However, since finite cyclic groups have no total order that respects the group structure, care should be taken when using the `Ord` operators. In particular, as an example, given field elements `x`, `y`, `z` and the implemented order `x > y`, it is not guaranteed that `x + z > y + z`. However, since the Ord operators are not used in the code, we could not identify any issues related to this implementation.

Compression Functions

We reviewed the implementation of the `CryptographicHash`, `Permutation`, and `Compression` traits in the folder `symmetric` as well as the hash functions `Blake3`, `Rescue`, `Poseidon`, `Poseidon2`, `Monolith`, and `Keccak`.

Since the specification of `Poseidon2` [GKS23] only shows that the matrix

$$\begin{bmatrix} 5 & 7 & 1 & 3 \\ 4 & 6 & 1 & 1 \\ 1 & 3 & 5 & 7 \\ 1 & 1 & 4 & 6 \end{bmatrix}$$

is maximum distance separable (MDS) for primes p of size larger than `2^32`, our team asked the Polygon team to explicitly check the MDS property for this matrix over the `Mersenne31` prime. The Polygon team subsequently informed us that they did, in fact, check this property using a custom script.

Our team found that in `Poseidon2`, the 'round constants' are initialized using Rust's `rand` crate, while `Poseidon2`'s protocol description [GKS23] assumes that those constants are initialized using the 'Grain LFSR' mechanism (referenced on page 8). However, we could not identify issues related to this deviation.

Keccak Air

For the constraint system of the `Keccak` function, we compared its implementation against a native implementation and examined the padding used to make the trace a power of two. We could not identify any issues. During our review, the Polygon team identified a missing constraint on the input data (see the fix in this PR). We found that there is no test that checks Keccak Air's output against a native

implementation. However, the Polygon team stated that the code is adapted from a different, thoroughly tested codebase with only minor changes being made.

Commitment Scheme and Mixed Matrix Merkle Tree (MMMT)

For the implementation in `\matrix`, we did not identify any issues. However, our team found that most of the functions assume that the number passed as input is a power of two. If possible, we suggest passing in the `base_2` logarithm of the input as the parameter to ensure type-safeness (Issue D).

The code contains traits for standard polynomial commitment schemes (PCS) as well as mixed matrix commitment schemes (MMCS) (described here). We identified two missing checks in the folder `\commit` (Issue E). The MMCS implementation in the folder `merkle-tree` allows the building of authenticated data structures for matrices of different sizes that generalize Merkle trees by including data not only at the leaf level but on any level except the root. We did not find any issues in this part of the code. In particular, we reviewed the code against second preimage attacks for Merkle Trees but found the code robust against such an attack due to the usage of different hash and compress functions on the leafs and higher layers.

Additionally, our team found that the implementation of the `circle` STARK polynomial commitment scheme [HLP24] in the file `circle/pcs.rs` is incomplete and work-in-progress at the time of this audit. We reviewed the code against [HLP24] and found that the PCS is missing a proper implementation of the functions `open` and `verify`. Although we did not find any issues, we recommend implementing a proper handling of the point at infinity for the trait `PolynomialSpace` (Suggestion 2).

Discrete Fourier Transformations (DFTs)

We examined the implementation of Discrete Fourier Transformations (DFTs) in the folders `\mds` and `\dft` and did not identify any issues. In particular, we compared the implementation of various FFT algorithms against their specifications (such as [BLD+10]). Each function is well-tested — in particular, each implementation of FFT is compared against the results of a naive DFT.

In addition, we reviewed the implementation of the barycentric interpolation in the folder `\interpolation` and compared it to the linked specification. We found one missing check on the input (Issue E).

STARK

We investigated the implementation of the STARK prover system (see [BBH+18]) in the folder `\uni-stark`. We found that the verifier is missing sanity checks on certain parameters (Issue A). In addition, we found that the verifier is using assertions instead of error messages, which could be exploited for a denial of service (DoS) attack (Issue F, Suggestion 1).

We compared the FRI protocol against the Multi-FRI paper. This paper describes a variant of FRI that allows the handling of polynomials of varying degrees. In the code, we found that the collinearity check is only done in the last round, while the standard FRI protocol (as explained in [BBH+18]) and the protocol in the Multi-FRI paper perform this check for each round. The Polygon team informed our team that they are aware of this deviation. Our team further investigated this deviation and could not identify any issues. However, we are unable to reason about the security impact in the light of Fiat-Shamir round security ([BGK+23]).

In addition, we found a subtle deviation in the code from the specification described in the Multi-FRI paper. The Polygon team informed our team that they are aware of this deviation from the specification. Our team further investigated this deviation and could not identify any issues. We recommend adopting the paper and its security proof (Suggestion 6).

Moreover, we found that the FRI verifier currently lacks robustness against out-of-boundary attacks (Issue E).

According to [FFK+23], FRI-based protocols produce non-malleable proofs. As a result, our team did not reason about associated attacks. Moreover, since the code is currently non-hiding, our team did not reason about zero-knowledge security.

The low-degree test of DEEP polynomials is not done explicitly, but it is assumed that the FRI polynomial commitment scheme assures that the quotient is in proximity to a low-degree polynomial. We could not identify any issues with this approach.

We reviewed the code against best practice Fiat-Shamir principles (see [BPW16] and [BGK+23]) and found that the Fiat-Shamir challenger is not initialized properly (Issue B). In addition, the final value in the commit phase of the FRI protocol is not absorbed into the challenger even though query point challenges are generated afterwards (Issue C).

We analyzed the Fiat-Shamir challenger implementation against best practices ([AKM+23]) and could not identify any issues.

Furthermore, we did not identify any issues in the implementations of the folder `\air`. For the implementation of PAIR (pre-processed AIR), we found that it is not currently used in the codebase.

Code Quality

We performed a manual review of the repositories in scope and found the codebases to be generally organized and well-written. However, we found that the implementation uses assertions to verify the correctness of input data or computations, while our team recommends using error messages instead to allow the user to benefit from a more graceful handling of such cases (Suggestion 1).

Tests

The repositories in scope include sufficient test coverage.

Documentation and Code Comments

The implementations generally have publicly accessible sources. The only exception our team identified was with the Multi-FRI paper provided for this review, which was still an early draft that was not yet made public and also consisted of a few imprecisions that our team shared with the Polygon team. Additionally, we found that code comments sufficiently describe the intended behavior of security-critical components and functions.

Scope

The scope of this review was sufficient. However, the Polygon team excluded certain folders (`/brakedown`, `/code`, `/lde`, `/reed-solomon`) from the audit, and our team noted that `circle-FRI` has not been completed yet. We recommend performing a comprehensive, follow-up audit once the codebase is updated — that is, after the addition of a recursive layer and, if possible, the hiding property, as well as the completion of `circle-FRI`. Note that the accompanying documentation, such as the Multi-FRI paper, was not reviewed by our team.

Dependencies

We did not identify any vulnerabilities in the implementation's use of dependencies.

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: Uni-Stark Verifier Trusts the Prover on the Validity of Proof Parameters	Unresolved: In Discussion
Issue B: Fiat-Shamir Initialization Is Incomplete	Partially Resolved
Issue C: Fiat-Shamir in FRI Is Incomplete	Resolved
Issue D: Missing Checks on the Validity of Inputs	Partially Resolved
Issue E: The FRI-Verifier Is Vulnerable to Multiple Buffer-Overflow Conditions	Partially Resolved
Issue F: Rust Assert Checks Could Potentially Crash the Verifier Strategically	Partially Resolved
Suggestion 1: Return Error Messages Instead of Panics	Partially Resolved
Suggestion 2: Improve Handling of the Point at Infinity in the Trait PolynomialSpace	Unresolved
Suggestion 3: Clarify Contradiction Around the Function from_base_slice (Known Issue)	Resolved
Suggestion 4: Implement a Script To Compute the Soundness as a Function of All Relevant Parameters	Partially Resolved
Suggestion 5: Allow Custom Global Domain Separators in Hash Functions	Partially Resolved
Suggestion 6: Update Multi-FRI Paper	Unresolved

Issue A: Uni-Stark Verifier Trusts the Prover on the Validity of Proof Parameters

Location

`uni-stark/src/verifier.rs`

Synopsis

The verifier does not check the validity of certain proof parameters (`degree_bits` and the number of the FRI commitments) but, instead, trusts the prover to provide the correct data.

Impact

A malicious prover could send an incorrect value to `degree_bits` or manipulate the number of the FRI commitments, which could lead to undefined behavior. For example, by pushing arbitrary data to the vector of FRI commitments, the prover can potentially trick the verifier into using an incorrect generator for the domain.

Technical Details

At the moment, the prover sends a proof consisting of commitments, opened values, an opening proof, and the parameter `degree_bits`. A check on the valid shape is performed for the opened values. However, a check is missing on the validity of `degree_bits` and the number of the FRI commitments contained in the opening proof. These should be verified against the AIR itself.

Remediation

We recommend implementing checks on the relevant proof parameters in the verifier by comparing them against values derived from the associated AIR.

Status

The Polygon team has acknowledged the finding but stated that they do not consider it to be a security issue since they refer to `degree_bits` as the source of truth, which cannot be maliciously changed in any successful attack because if a Merkle proof length does not match `degree_bits` (considering any adjustments based on blowup, etc.), it is the Merkle proof that is at fault. The Polygon team added that `FieldMerkleTreeMmcs` should ensure that the expected dimensions provided by the caller match the Merkle proof length. Additionally, the team does not consider adaptations of any associated Merkle proofs to the wrong `degree_bits` as a possibility.

Verification

Unresolved: In Discussion.

Issue B: Fiat-Shamir Initialization Is Incomplete

Location

`uni-stark/src/prover.rs`

`uni-stark/src/verifier.rs`

`keccak-air/examples`

Synopsis

The initialization of the Fiat-Shamir challenger for the univariate STARK implementation does not adhere to the principles of Fiat-Shamir, which require the challenger to absorb all information the verifier has access to at any given step in the computation.

Impact

Not absorbing parameters such as the public inputs, the FRI configuration, or the degree of the polynomial allows a malicious prover to tamper with this data. Depending on the use case of this STARK implementation, this can lead to security-related consequences (see, for example, [DMW+23]).

Technical Details

In the current implementation, the prover and the verifier initialize the challenger with an empty vector. The principles of Fiat-Shamir recommend absorbing all the information the verifier has access to. Based on this, we identified that the following items are missing:

- A global domain separator to ensure that proofs of different domains are incompatible;
- A configuration for FRI, including the `log_blowup`, `num_queries`, `proof_of_work_bits`, and a representation of the Mixed-Matrix Commitment Scheme (MMCS);
- A representation of the underlying Fields and Field Extensions;
- All pre-decided generators;
- A representation of the AIR in use;
- A representation of the Polynomial Commitment Scheme (PCS);
- The public values used;
- The potential maximal degree of all constraints; and
- The number of atomic constraints — that is, the number of elements in the `enum SymbolicExpression`.

Remediation

We recommend properly initializing the challenger by including all of the items listed above.

Status

The Polygon team has added the public values and the trace degree. For the other data, the team stated that a domain separator should be sufficient.

Verification

Partially Resolved.

Issue C: Fiat-Shamir in FRI Is Incomplete

Location

`fri/src/prover.rs#L27`

Synopsis

The Fiat-Shamir challenger does not currently absorb the value `final_poly`, which is the result of the commit phase in the FRI protocol.

Impact

While our team did not reason about a specific attack, incorrect Fiat-Shamir implementations have been shown to have security-relevant consequences (see, for example, [DMW+23]).

Technical Details

During FRI, the prover divides and folds a polynomial in order to reduce its degree and to prove proximity to a low-degree polynomial. After each folding round, the prover absorbs the commitment hash on the polynomial of that round. The value `final_poly` is the result of the commit phase in the FRI protocol and is currently not absorbed into the challenger.

Since the challenger is used to sample the query indices after the commit phase, and the verifier sees the value `final_poly`, following the principles of Fiat-Shamir, the prover has to absorb `final_poly` into the challenger before squeezing the query indices.

Remediation

We recommend absorbing the value `final_poly` into the challenger before squeezing the query indices.

Status

The Polygon team has implemented the remediation as recommended (PR #393).

Verification

Resolved.

Issue D: Missing Checks on the Validity of Inputs

Location

`field/src/field.rs#L304`

`commit/src/domain.rs#L91`

`matrix/src/bitrev.rs#L28`

`interpolation/src/lib.rs#L46`

Synopsis

Our team identified a number of missing checks on the validity of inputs.

Impact

A missing check on relevant input data can lead to crashes or undefined behavior in the code.

Technical Details

We identified the following missing checks:

- In the file `commit/src/domain.rs`, the function `split_domains` panics if `log_n` is smaller than `log_chunks`. It should be checked that this is not the case. In addition, the code fails if `num_chunks` is not a power of two.
- In the same file, in the function `selectors_on_coset`, it is implicitly assumed that the inverses of denoms for the `single_point_selector` exist. This should be checked with `assert_neq!(coset.shift, Val::one())`.
- In the file `matrix/src/bitrev.rs`, the function new panics if `inner.height` is not a power of two due to the function `log2_strict_usize`. This occurs in several other locations, including in the folder `matrix` and in other files (more specifically, searching the function in the codebase yields 76 results in 24 different files).
- In `interpolation/src/lib.rs`, the function `interpolate_coset` panics if the point passed as an input is one of the roots of unity — that is, equals `subgroup_i`. In this case, the inverse does not exist. It should be verified that this is not the case.
- In the file `field/src/field`, the function `monomial` crashes for exponents larger than the extension degree `D`. It should be checked that the exponent is smaller than the extension degree `D`.

Remediation

We recommend implementing the relevant checks listed above. Additionally, and as noted in Suggestion 1, we recommend returning error messages instead of panics for these checks.

Furthermore, we recommend making the code more robust and typesafe, whenever possible. Our team recognizes that for most functions in the codebase, it is a precondition that certain input data be powers of two (such as the height of matrices). One possible mitigation is passing in `log2` of a number instead of the number. In addition, it could be stated as a precondition in the respective modules.

Status

The Polygon team has introduced changes to clarify assumptions about valid inputs (PR #387). The team also noted that they could not identify any instances where these panics could realistically be triggered; however, they have added a warning in the `README`, in case such instances are present but were undetected. The Polygon team additionally stated that if one conservatively assumes a malicious prover could trigger a panic, there are reasonable ways for users to make their applications resilient (for example, by implementing auto-restarts).

Verification

Partially Resolved.

Issue E: The FRI-Verifier Is Vulnerable to Multiple Buffer-Overflow Conditions

Location

`fri/src/verifier.rs`

Synopsis

The verifier currently lacks robustness against out-of-boundary attacks, such as buffer, slice, or array overflows, and does not perform several critical checks on the shape of the prover's data.

Impact

A malicious prover could potentially push irrelevant data to the end of certain vectors in the proof, which could lead to unpredictable behavior or cause the code to crash. Crashes could open an attack vector for denial of service (DoS) types of attacks. For example, a longer than expected `commit_phase_commits` vector would imply an unexpected generator.

Technical Details

We identified the following missing boundary checks, which should:

- Verify the length of `commit_phase_commits` (computed by the verifier) against a number (see Issue A);
- Ensure that the `commit_phase_openings` length equals `log_max_height`;
- Confirm the correct length of `reduced_openings`; and
- Check the data type of reduced openings in the `verify_challenges` function.

Remediation

We recommend implementing the proper checks.

Status

The Polygon team has made modifications to clarify assumptions about boundary conditions (PR #387). The team also noted that they could not identify any realistic scenarios where these crashes could be triggered; however, they have included a warning in the `README` in case such instances are present but were undetected. In addition, safe Rust's bound check currently prevents buffer overflows, and, hence, the only prevalent risk is that the code panics. The Polygon team additionally stated that assuming a

malicious prover might be able to trigger a panic, users can adopt reasonable measures to enhance the resilience of their applications (for example, by implementing auto-restarts).

Verification

Partially Resolved.

Issue F: Rust Assert Checks Could Potentially Crash the Verifier Strategically

Location

`fri/src/verifier.rs`

Synopsis

In the verifier, multiple checks are implemented as Rust's `assert!()` macros, thus resulting in the code crashing instead of recovering.

Impact

A malicious prover could intentionally crash the code to open an attack vector for performing DoS attacks.

Remediation

We recommend implementing proper error handling, such that the verifier returns an `INVALID_PROOF` error instead of crashing.

Status

The Polygon team has implemented updates to clarify assumptions regarding Rust assertion checks (PR #387). The team also noted that they could not identify any realistic situations where these checks could panic; however, they have added a warning in the `README`, in case such instances are present but were undetected. The Polygon team additionally stated that, assuming a malicious prover could potentially trigger a panic, users can take sensible precautions to make their applications more robust (for example, by setting up auto-restarts).

Verification

Partially Resolved.

Suggestions

Suggestion 1: Return Error Messages Instead of Panics

Location

Examples (non-exhaustive):

`matrix/src/bitrev.rs#L28`

`interpolation/src/lib.rs#L46`

`field/src/field.rs#L304`

commit/src/domain.rs#L91

baby-bear/src/baby_bear.rs#L303

baby-bear/src/extension.rs#L27

commit/src/domain.rs#L124-L125

Synopsis

At various instances in the code, assertions verify the validity of input data or the correctness of a computational result. In case of incorrect data, the code panics. Since Plonky3 is a toolkit, the extent of its usability may not be immediately clear for users. As a result, users would benefit from a more graceful handling of such cases.

Mitigation

We recommend adding error messages to make the code more robust against failures.

Status

The Polygon team has implemented updates to clarify assumptions regarding error handling (PR #387).

Verification

Partially Resolved.

Suggestion 2: Improve Handling of the Point at Infinity in the Trait PolynomialSpace

Location

circle/src/util.rs#L77

circle/src/domain.rs

Synopsis

The implementation of the trait `PolynomialSpace` for the `CircleDomain` interprets None as the point at infinity. This can lead to confusion and security issues. For example, at the moment, a potential conflict might arise with the function `next_point` since it returns None for the non-standard case.

Mitigation

We recommend implementing the projective line as a type, thereby addressing the point at infinity properly.

In addition, and to avoid confusion with the function `next_point`, we further recommend changing the return type of this function from `Option<Field>` to `Result<Option<Field>, Err>`.

Status

The Polygon team has acknowledged this suggestion and noted that since it is a minor one, the recommended mitigation will not be implemented at this time due to time limitations, but will be taken into consideration for future releases.

Verification

Unresolved.

Suggestion 3: Clarify Contradiction Around the Function from_base_slice (Known Issue)

Location

`field/src/field.rs#L283`

`src/extension/binomial_extension.rs#L519`

Synopsis

The comment of the function `from_base_slice` in the trait `AbstractExtensionField` suggests that the input to this function can be a slice of length at most `D`. However, the implementation for `BinomialExtensionField` panics if the slice is not of length exactly `D`.

Mitigation

We recommend resolving the contradiction by either updating the comment or rewriting the implementation.

Status

The Polygon team has implemented the mitigation as recommended (PR #382).

Verification

Resolved.

Suggestion 4: Implement a Script To Compute the Soundness as a Function of All Relevant Parameters

Synopsis

The soundness of STARK and FRI in particular depend on many parameters, such as numbers of queries, field size, extension degree, etc. For users to target a particular soundness level, a formula or script is needed that computes this number as a function of all adjustable parameters.

Mitigation

We recommend implementing a script to compute the system's soundness as a function of all configurable parameters.

Status

The Polygon team has addressed a conjectured soundness (PR #387); however, they stated that proven soundness would be more challenging to compute in Rust.

Verification

Partially Resolved.

Suggestion 5: Allow Custom Global Domain Separators in Hash Functions

Location

`rescue/src/rescue.rs#L80`

Synopsis

Since Plonky3 is a toolkit that might be used in different systems on multiple chains, customizable global domain separators for all hash functions might be needed.

Mitigation

We recommend implementing a script that users can run to customize all hash functions with global domain separators.

Status

The Polygon team has acknowledged this suggestion but stated that this would require users calling `observe(domain_separator)` after constructing a `DuplexChallenger` or another challenger. The team additionally stated that it may not be necessary to change any library code to do so, as it may be sufficient to simply add some comments or documentation to encourage the practice.

Verification

Partially Resolved.

Suggestion 6: Update Multi-FRI Paper

Synopsis

We found a subtle deviation in the code from the specification described in the Multi-FRI paper (unpublished draft by A. Green and D. Vagner). In particular, the Fiat-Shamir randomness is used differently in the code than in the Multi-FRI paper. In the commit phase, a random value is applied earlier in the code than in the paper for the `roll` operator to polynomials of degree `d_i`. The `roll` operator takes a random `alpha^r` for round `r`, but the code already performs this step in the initialization step for all polynomials. With regards to the paper, it takes `alpha^0` instead of `alpha^r` to `roll` polynomials in round `r`.

Mitigation

Although our team further investigated this deviation and could not identify any issues, we still recommend adopting the paper and its security proof to address this inconsistency.

Status

The Polygon team is aware of this subtle deviation from the specification and agrees that the code should be updated. However, the team added that this cannot be implemented in a short time frame and stated that they are nevertheless committed to updating it before the paper is published.

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.