Least Authority

PRIVACY MATTERS

gnark

Security Audit Report

Consensys Software, Inc.

Final Audit Report: 20 September 2024

Table of Contents

Overview

Background

Consensys Software, Inc. has requested that Least Authority perform security audits of the Linea prover and cryptography libraries. This audit is Phase 1 and focuses specifically on the `gnark` library.

Project Dates

- **June 5, 2024 - July 1, 2024:** Initial Code Review *(Completed)*
- **July 3, 2024:** Delivery of Initial Audit Report *(Completed)*
- **September 19, 2024:** Verification Review *(Completed)*
- **September 20, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- 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 `gnark` cryptographic libraries 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:

- `gnark`:
  - https://github.com/Consensys/gnark/tree/master/std/gkr
  - https://github.com/Consensys/gnark/tree/master/std/polynomial
  - https://github.com/Consensys/gnark/tree/master/std/sumcheck
  - https://github.com/Consensys/gnark/tree/master/constraint (all go files)
  - https://github.com/Consensys/gnark/tree/master/constraint/bn254
  - https://github.com/Consensys/gnark/tree/master/constraint/solver
- `gnark-crypto`:
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377(all go files)
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fp/hash_to_field
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr(all go files)
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr/fft
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr/hash_to_field
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr/iop
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr/mimc
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr/polynomial
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/fr/sumcheck
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/internal/fptower
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bls12-377/kzg
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bns254 (all go files)
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bns254/fp/hash_to_field
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr (all go files)
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr/fft
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr/hash_to_field
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr/iop

  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr/mimc
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr/polynomial
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/fr/sumcheck
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/internal/fptower
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bn254/kzg
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761 (all go files)
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fp/hash_to_field
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr (all go files)
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr/fft
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr/hash_to_field
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr/iop
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr/mimc
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr/polynomial
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/fr/sumcheck
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/internal/fptower
  - https://github.com/Consensys/gnark-crypto/tree/master/ecc/bw6-761/kzg

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

- `gnark`: 02d4678924927050149ccda18dce6133a15601ff
- `gnark-crypto`: 564b6f724c3beac52d805e6e600d0a1fda9770b5

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

- `gnark`:
  https://github.com/LeastAuthority/Consensys-gnark
- `gnark-crypto`:
  https://github.com/LeastAuthority/Consensys-gnark-crypto

For the verification, we examined the Git revision:

- `gnark`: 3a0fa0f316437854d56bf10a1e75811df9697f46
- `gnark-crypto`: 703a260c2f991d01e245adf53d20f76af4210c5f

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://linea.build
- Linea prover/cryptography audits: request for proposals (Google Doc) *(shared with Least Authority via email on 8 March 2024)*
- Linea Prover audit plan Q1 2024 (Google Doc) *(shared with Least Authority via email on 11 April 2024)*

In addition, this audit report references the following documents:

- G. Adj and F. Rodríguez-Henríquez, "Square root computation over even extension fields." *IACR Cryptology ePrint Archive*, 2012, [AR12]

- A. Belling, A. Soleimanian, and O. Begassat, “Recursion over Public-Coin Interactive Proof Systems; Faster Hash Verification.” *IACR Cryptology ePrint Archive*, 2022, [BSB22]
- J. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, et al., “High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves.” *IACR Cryptology ePrint Archive*, 2020, [BGM+20]
- Q. Dao, J. Miller, O. Wright, and P. Grubbs, “Weak Fiat-Shamir Attacks on Modern Proof Systems.” *IACR Cryptology ePrint Archive*, 2023, [DMW+23]
- S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, “Delegating Computation: Interactive Proofs for Muggles.” *Microsoft*, 2008, [GKR08]
- R. Granger and M. Scott, “Faster Squaring in the Cyclotomic Subgroup of Sixth Degree Extensions.” *IACR Cryptology ePrint Archive*, 2009, [GS09]
- K. Karabina, “Squaring in Cyclotomic Subgroups.” *IACR Cryptology ePrint Archive*, 2010, [Karabina10]
- K. Rubin and A. Silverberg, “Compression in Finite Fields and Torus-Based Cryptography.” *Society for Industrial and Applied Mathematics (SIAM),* 2008, [RS08]
- N. Stephens-Davidowitz, “Ring-SIS and Ideal Lattices.” *Massachusetts Institute of Technology*, 2018, [S18]
- J. Thaler, “Proofs, Arguments, and Zero-Knowledge.” *Georgetown University*, 2023, [Thaler23]
- J. Thaler, “The GKR Protocol and Its Efficient Implementation.” *Georgetown University*, 2017, [Thaler17]
- J. Zhang, T. Liu, Y. Horesh, W. Wang, Y. Zhang, et al., “Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time.” *IACR Cryptology ePrint Archive*, 2020, [ZLH+20]
- Investigation report on the use of SIS to build fast SNARKs: https://hackmd.io/7OODKWQZRRW9RxM5BaXtIw

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks;
- Denial of service (DoS) attacks and security exploits that would impact the implementation or disrupt its execution;
- Vulnerabilities within individual components and whether the interaction between the components is secure;
- Exposure of any critical information during interaction with any external libraries;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other methods of exploitation;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

OuOur team performed a security audit of the GKR protocol implementation of Linea. GKR is an interactive proving protocol, which is made non-interactive through the Fiat-Shamir heuristic. The Linea team has also written a paper [BSB22] about their usage of GKR, containing a protocol description of GKR and an analysis on the usage of the Fiat-Shamir heuristic in this setting.

In addition, we audited low-level building blocks, such as base and scalar fields of the elliptic curves in `gnark` cryptographic libraries. The `gnark` project implements an open-source, low-level cryptographic library written in Go that provides tools for working with zk-SNARKs (zero-knowledge succinct non-interactive arguments of knowledge). While `gnark` offers high-level APIs to build circuits, the underlying cryptography is implemented in `gnark-crypto`.

System Design

Our team examined the design of the `gnark` cryptographic libraries and found them to be well-designed, with a strong emphasis on security.

We examined the various implementations of hash functions that map to either the base or the scalar field of the `bls12-377`, `bn254`, the `bw6-761` elliptic curves, as well as the `mimc` permutation. We analyzed the code against standard errors, such as high biases, overflows, or unsafe parameter choices and could not identify any issues.

Moreover, the Linea team has implemented hash to field functions for `bls12-377` and `bn254`, based on the Short Integer Solution (SIS) problem, which we analyzed and discussed with the Linea team in detail. Our team did not identify any issues relating to these areas of concerns.

We also investigated the extension field towers for those curves and checked whether pairings are generated properly by comparing the implementation against their specifications and identified an issue (Issue E).

Our team additionally reviewed the `fft` code and checked whether proper tests are in place. We analyzed their implementation of an Interactive Oracle Proof for a permutation and a quotient argument and identified an Issue (Issue D).
We assessed the implementation of the KZG polynomial commitment scheme. Assuming that the used randomness (Powers of Tau) is computed in a safe way, (e.g., by means of a proper multi-party computation), our team could not identify any issues.

Furthermore, we analyzed the GKR implementation in the Linea team's paper [BSB22] and the original GKR paper [GKR08] by reviewing the implementations in both `gnark` and `gnark-crypto` for the different elliptic curves, which differ only with regards to the use of the `claims` manager and different gate definitions. Our team could not identify any issues relating to these areas of concerns. However, we recommend writing a proper specification for these implementations of GKR since the underlying academic work [BSB22] has some inconsistencies and does not include all the details of the specification (Suggestion 2).

Our team also analyzed the various implementations of the `sumcheck` protocol operations performed through the GKR proving and verifying mechanism and the relevant topological sorting functions. We additionally thoroughly reviewed the use of the Fiat-Shamir heuristic, which renders the GKR proof non-interactive, and could not identify issues within the GKR functionality.

Code Quality

Our team found the code to be clean, well-organized, and in adherence to standard best practices for advanced cryptography, which we found to be considerately implemented.

Tests

The repositories in scope include sufficient test coverage.

Documentation and Code Comments

The documentation provided by the Linea team was broad and sufficient. Our team noted some ways to improve the documentation even further through updating the academic reference [BSB22] to the GKR implementation in regard to some inconsistencies (Suggestion 1) and adding a specification (Suggestion 2). Additionally, code comments contain some inaccuracies, which we recommend be corrected (Suggestion 3).

Scope

The scope of this review did not include the `fr/permutation/permutation.go` repository. Additionally, our team did not manually audit `internal/fptower/e2_amd64.s` but rather reviewed whether the Linea team implemented proper tests by comparing the system against a high level implementation.

Dependencies

Our team 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: Incorrect Computation of Negative Factorial	Resolved
Issue B: Incorrect Polynomial Evaluation for Domain Shifts Greater Than Five	Resolved
Issue C: Polynomial Addition Potentially Computes Unexpected Sum	Determined Non-Issue.
Issue D: Permutation NIZK Does Not Include FFT Generator in the Transcript	Unresolved
Issue E: Incorrect Conjugate in F_(p^3) for BW6-761, BW6-756 and BW6-633	Resolved
Suggestion 1: Update Academic Reference	Unresolved
Suggestion 2: Write Specification for GKR	Unresolved
Suggestion 3: Update Code Comments	Partially Resolved

Issue A: Incorrect Computation of Negative Factorial

Location

`std/sumcheck/lagrange.go#L7`

Synopsis

The implementation of the negative factorial function `negFactorial(n)` computes `negFactorial(1)=1`, but it should compute `negFactorial(1)=-1`. As a consequence, the zero'th Lagrange polynomial `L_0` is computed incorrectly for any linear (degree 1) Lagrange base.

Impact

The impact is low since linear interpolation occurs rarely – if at all – in targeted applications, such as zero-knowledge proving systems. However, in cases where linear interpolation would occur, the Issue will lead to the generation of false proofs.

Remediation

We recommend implementing a proper negative factorial function.

Status

The Linea team has resolved this Issue in PR1158.

Verification

Resolved.

Issue B: Incorrect Polynomial Evaluation for Domain Shifts Greater Than Five

Location

`fr/iop/polynomial.go#L184`

Synopsis

The function evaluates a polynomial at a given point. However, since the computation is dependent on the size of the associated shift factor of the polynomial, the function fails to initialize the `fft` generator g.

Preconditions

The shift factor of the polynomial would have to be larger than five.

Remediation

We recommend initializing g to be `fft.Generator(uint64(p.size))`.

Status

The Linea team will be resolving this Issue with PR539 by initializing g correctly.

Verification

Resolved.

Issue C: Polynomial Addition Potentially Computes Unexpected Sum

Location

`fr/polynomial/polynomial.go#L98`

Synopsis

For correct execution, the function `p.Add(p1, p2)` must check if p is equal to the larger polynomial `bigger` of p1 or p2. However, the function only checks whether the degrees and the constant terms of p and `bigger` are equal.

Impact

Without loss of generality, and assuming `deg(p1) < deg(p2)`, an attacker can use the missing checks to compute `p ← p + p1` for `p ≠ p2`, instead of the expected result `p ← p1 + p2`.

Preconditions

The constant terms of p and p2 would have to be equal.

Feasibility

Straightforward.

Remediation

We recommend implementing equality checks for all coefficients.

Status

During the verification phase, our team determined this finding to be a Non-Issue, as the check is on the pointer addresses and not the constant terms.

Verification

Determined Non-Issue.

Issue D: Permutation NIZK Does Not Include FFT Generator in the Transcript

Location

`fr/permutation/permutation.go#L135`

Synopsis

According to the principles of the strong Fiat-Shamir transformation [DMW+23], the FFT domain generator should be part of the initial Fiat-Shamir transcript.

Impact

Without an inclusion of the generator into the initial transcript, a malicious prover might be able to strategically change the generator as a function of the proof, the witness, and the Fiat-Shamir randomness to, for example, find a point where the argument is false, but the proof is correct (since soundness is not perfect). While we were not able to leverage this Issue to perform an attack, including the generator into the transcript is common practice.

Remediation

We recommend including the generator into the initial transcript before any randomness is derived.

Status

The Linea team stated that it is unlikely that the prover would be able to choose the FFT domain generator and have therefore decided not to fix this Issue. However, our team still recommends addressing this finding, as leaving this Issue unresolved could lead to a malleable situation.

Verification

Unresolved.

Issue E: Incorrect Conjugate in F_(p^3) for BW6-761, BW6-756 and BW6-633

Location

[`internal/fptower/e3.go#L128`]

Synopsis

The code implements incorrect conjugate functions in the degree three extension fields `F_(p^3)` for the elliptic curves `BW6-761`, `BW6-756` (out of scope), and `BW6-633` (out of scope).

Impact

None. The functions are not used anywhere in the codebase.

Technical Details

The implementation uses `-1` to compute the conjugate, while it should use the cubic non-residue `-4`, according to the specification of the associated `fp-towers`.

Remediation

We recommend either deleting or rewriting the conjugation functions.

Status

The Linea team has resolved this Issue by removing the conjugation functions in PR514.

Verification

Resolved.

Suggestions

Suggestion 1: Update Academic Reference

Location

Examples (non-exhaustive):

`std/gkr/gkr.go`

`fr/gkr/gkr.go`

Synopsis

The academic reference to the GKR implementation should include correct and precise information. Our team identified the following instances within the reference paper [BSB22] that need updating:

- In Figure 15 (p. 15): `V_0(\rho)` is not defined.
- In Figure 15 (p. 15): The claim register `claims` is only defined in a comment. Type definition could be made more direct.
- Definition B.2 (p. 14): The batch assignment should be defined for the specific cases of input and output gates.
- In Figure 15 (p. 15): The call to `miniProtocol2` should be on `(v, claim')`.

- Remark B.3 (p. 14): For the defining equation for `B(v)(x)` over `K^n`, the summation should be over the hypercube `{0,1}^n`.

Mitigation

We recommend updating the paper.

Status

The Linea team has acknowledged the suggestion and created a GitHub issue (#1280)to address the finding.

Verification

Unresolved.

Suggestion 2: Write Specification for GKR

Location

Examples (non-exhaustive):

`std/gkr/gkr.go`

`fr/gkr/gkr.go`

Synopsis

While an academic reference is a useful resource that developers and auditors can refer to, it does not serve the same purpose as a specification, which could contain more information about the extension field tower and be a necessary addition to this complex protocol.

Mitigation

We recommend considering writing a specification for Linea GKR.

Status

The Linea team has acknowledged the suggestion and created a GitHub issue (#1280) to address the finding.

Verification

Unresolved.

Suggestion 3: Update Code Comments

Location

Multiple locations throughout the codebase.

Synopsis

The following code comments contain inaccuracies:

- `std/sumcheck/lagrange.go#L25` should be updated to be `δ_i(at)` by noting it is equal to `δ_{i-1}(at) × (r-i+1) × (r-i)`$^{-1}$ `× i`$^{-1}$ `× (-len(values)+i)`.
- The definition of the Ring in `fr/sis/sis.go#L72` should be updated to `Z_p[X]/X^d + 1`.
- In `fr/iop/ratios.go#L55`, `(Π_{k<j}Π_{i<m}(β-P`$_i$`(ω`^{k}`)))/(β-Q`$_i$`(ω`^{k}`))` should be updated to `Π_{k<j}Π_{i<m} (β-P`$_i$`(ω`^{k}`))/(β-Q`$_i$`(ω`^{k}`))`.

- In `internal/fptower/e2.go#L104`, as well as in similar locations in other curves, the comment should be updated to describe the correct functionality.
- `fr/polynomial/multilin.go#L158` should be updated to `res <- 1 + 2 * qᵢ * hᵢ - qᵢ - hᵢ`.
- The extension field tower in `ecc/bw6-761/bw6-761.go#L18ecc/bw6-761/bw6-761.go#L17` should be defined as `F_(p^3)[x]/<v²-u>`.
- In `internal/fptower/e2.go#L43`, the comment should be updated to warn users that this function does not implement an order in finite fields that respects the field structure. (However, our team notes that this function is not used anywhere in the codebase.)

Mitigation

We recommend updating the comments to prevent misunderstandings and security risks.

Status

The Linea team has resolved part of this suggestion (the fourth and sixth bullet point) with PR511.

Verification

Partially Resolved.

About Least Authority

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

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.