Mysten Fastcrypto Groth16 Audit

Shresth Agrawal[1,2] Pyrros Chaidos[1,3]

[1] Common Prefix
[2] Technical University of Munich
[3] University of Athens

October 19, 2023
Last update: October 24, 2023

1 Overview

1.1 Introduction

Mysten Labs commissioned Common Prefix to audit the Groth16 verification implementation within their fastcrypto library. The primary objectives of the audit were to assess the security, adherence to the relevant publications, and also investigate performance optimizations, and code quality improvements to these particular implementations. Fastcrypto is a Rust-based library that implements selected cryptographic primitives and also serves as a wrapper for several carefully chosen cryptography crates, ensuring optimal performance and security for Mysten Labs' software solutions, including their blockchain platform, Sui.

Groth16 [1] is a zero-knowledge proof system that is optimized for small proof size and fast verification speed. A zero-knowledge proof system allows one to prove that a certain statement is true without disclosing the reason. Groth16 uses pairing-based curve cryptography and requires a trusted setup that is specific to each family of statements (effectively, a separate setup for each application). Such a proof system usually consists of three parts (mostly with considerable overlap in the codebase): (a) the setup, or generator, which produces the structured reference string (SRS) necessary to use the proof system; (b) the prover, who uses the SRS and some (potentially private) data (the *witness*) and produces a proof π that a statement s is true; and (c) the verifier, who using the SRS can verify π to check whether s is true without having to see the rest of the supporting data. This audit is limited to the verifier portion of the codebase.

Fastcrypto's Groth16 implementation relies on several Arksworks crates, including ark-bn254, ark-crypto-primitives, ark-groth16, etc., for the logic of the core proof system itself as well as for the underlying curve point, scalar and field element operations. The fastcrypto part of the code is for the most part tasked with optimizing the use of prepared verification keys, including serialization, deserialization, and conversion functionality.

This audit report evaluates the Groth16 verification implementation within the fastcrypto library. We have audited the code for security, efficiency, and reliability. The scope of this audit was limited to the implementation of the verification functionality of Groth16 and the required conversion and pre-processing functions within the Fastcrypto library and did not extend to the library's dependencies or other components. In particular, we note that the audit does not cover any of the Arkworks codebase. Additionally, the downstream applications in which the verification functionality is utilized, any code used for SRS generation, and proof generation lie outside the scope of this audit.

1.2 Audited Files

1. [eb77c755] fastcrypto-zkp/src/bn254/api.rs
2. [eb77c755] fastcrypto-zkp/src/bn254/verifier.rs

1.3 Disclaimer

This audit does not give any warranties on the bug-free status of the given code, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. This audit report is intended to be used for discussion purposes only. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the project.

The scope of the audit was constrained exclusively to the Fastcrypto wrapper code, with no examination conducted on its associated dependencies. Notably, the Arkworks Groth16 crate, upon which Fastcrypto is reliant, has not undergone a prior audit. Furthermore, the audit does not encompass any reference string generation functionality in terms of code or execution.

1.4 Executive Summary

Overall, the Groth16 wrapper and conversion functions are of very high quality, adhering to Rust's best practices.

The implementation relies on the upstream Arkworks implementation for most complex operations performed in the context of Groth16 verification, providing a wrapper functionality to utilize the relevant functions in fastcrypto. Due to this, the findings of the audit are mostly focused on simplicity and consistency of the codebase.

One thread of findings concerns the multiple functions that can be used to verify a proof (I-04) and the possibility of using them in a way that bypasses length checks (L-01).

The other findings suggest small changes to enforce consistency with the upstream codebase (I-03), internal consistency of serialization and deserialization (I-01) and finally using rust convention traits to align with language conventions and make the code structure clearer (I-02).

1.5 Findings Severity Breakdown

The findings are classified under the following severity categories according to the impact and the likelihood of an attack.

Level	Description
High	Logical errors or implementation bugs that are easily exploited. In the case of contracts, such issues can lead to any kind of loss of funds.
Medium	Issues that may break the intended logic, are deviations from the specification, or can lead to DoS attacks.
Low	Issues harder to exploit (exploitable with low probability), can lead to poor performance, clumsy logic, or seriously error-prone implementation.
Informational	Advisory comments and recommendations that could help make the codebase clearer, more readable, and easier to maintain.

2 Findings

2.1 High

None Found.

2.2 Medium

None Found.

2.3 Low

L01: Missing length validation in `verify_groth16`.

Affected Code: src/bn254/api.rs (line 54)

Summary: The `verify_groth16` function lacks length validation for the `proof_public_inputs_as_bytes` parameter. This is in contrast to the `verify_groth16_in_bytes` function, which does perform the check. Thus, in the intended flow of `verify_groth16_in_bytes` calling `verify_groth16` the check will be performed, but it will be skipped if `verify_groth16` is called directly.

Suggestion: It is recommended to move the length validation check, currently present in the `verify_groth16_in_bytes` function, to the `verify_groth16` function.

Status: Resolved [731d84c5]

2.4 Informational

I-01: Inconsistency in the serialization/deserialization functions for `PreparedVerifyingKey`

Affected Code: src/bn254/verifier.rs (lines 39,73)

Summary: The output type of the serialization function doesn't match the input type of the deserialization function. Therefore the output of the serialization operation cannot be directly passed as an argument to the deserialization function.

Suggestion: It is advisable to harmonize these functions for consistent usage.

Status: Resolved [f880626]

I-02: Consider using conversion traits instead of specialized functions.

Affected Code: src/bn254/verifier.rs (lines 108,142)

Suggestion: Consider implementing the conversion traits `From<ArkPreparedVerifyingKey>` for `PreparedVerifyingKey` and `From<PreparedVerifyingKey>` for `ArkPreparedVerifyingKey` instead of relying on specialized functions such as `as_arkworks_pvk` and `process_vk_special`.

Status: Resolved [99a771d, 1ab759c]

I-03: Maintain consistency with upstream code during point negation.

Affected Code: src/bn254/verifier.rs (lines 146,147)

Summary: In the Arkworks implementation, during the `prepare_verifying_key` operation, the `gamma_g2_neg_pc` and `delta_g2_neg_pc` affine points are temporarily converted to projective representation before negation, and are converted back to affine. The fastcrypto implementation omits these additional conversions. Even though these conversions are redundant, it may be beneficial to add those for consistency between the implementations in the event that future changes in the underlying libraries cause the results to differ.

Suggestion: Consider aligning the code with the conventions for conversion used in Arkworks.

Status: Acknowledged

I-04: Improve proof verification API

Affected Code:

- src/bn254/api.rs (lines 30,54)
- src/bn254/verifier.rs (line 154)

Summary: Currently, there are three functions responsible for proof verification: `verify_groth16_in_bytes`, `verify_groth16`, and `verify_with_processed_vk`.

Suggestion: For stylistic consistency, it is suggested to relocate the `verify_with_processed_vk` function to the `api.rs` file and call it from within the `verify_groth16` function.

Status: Resolved [04a098a, 99a771d]

References

1. J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.