Mysten ECVRF and Ristretto255 Audit

Shresth Agrawal[1,2] Petros Angelatos[1,3]
Pyrros Chaidos[1,4] Dionysis Zindros[1,5]

[1] Common Prefix
[2] Technical University of Munich
[3] National Technical University of Athens
[4] University of Athens
[5] Stanford University

April 07, 2023
Last update: April 15, 2023

1 Overview

1.1 Introduction

Mysten Labs commissioned Common Prefix to audit the Elliptic Curve Verifiable Random Function (ECVRF) and Ristretto255 implementations within their fastcrypto library. The primary objectives of the audit were to assess the security, adherence to the relevant RFCs, 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.

Verifiable Random Functions (VRFs) can be used to provide trustless randomness in blockchain protocols [2,4]. Elliptic Curve VRFs (ECVRFs) employ the elliptic curve Diffie–Hellman assumption for the security of the VRF [6]. The fastcrypto ECVRF implementation follows the draft-irtf-cfrg-vrf-15 RFC [5], except for its usage of the Ristretto255 curve. This deviation is due to the Ristretto technique, which offers advantages, such as constructing prime-order elliptic curve groups with non-malleable encodings, resulting in improved security properties [3]. The Ristretto255 implementation is a wrapper around the curve25519-dalek-ng implementation which follows the draft-irtf-cfrg-ristretto255-decaf448-07 RFC [3].By leveraging Ristretto255, Mysten Labs seeks to strengthen the ECVRF implementation while maintaining close compatibility with

the RFC. The company plans to utilize the fastcrypto ECVRF implementation to provide a reliable source of randomness for both the consensus and execution layers of their Sui platform.

This audit report comprehensively evaluates the ECVRF and Ristretto255 implementations within the fastcrypto library. The findings are categorized by severity, accompanied by proposed solutions for each identified issue. We have audited the code for security, efficiency, and reliability. The scope of this audit was limited to the ECVRF and Ristretto255 implementations and did not extend to the library's dependencies or other components.

1.2 Audited Files

1. [963205c6] vrf.rs
2. [963205c6] ristretto255.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.

1.4 Executive Summary

Overall the implementation of the ECVRF using the Ristretto255 is of very high quality, closely following the RFC guidelines and adhering to Rust's best practices.

No security-critical issues were identified during the audit process. However, minor deviations from the RFC specification were observed, such as the `ecvrf_encode_to_curve` function hashing twice instead of once as specified by the RFC, the `verify_output` function not being entirely in line with the RFC, and a discrepancy in Ristretto scalar deserialization leading to multiple representations for elements that should be unique.

Performance optimization suggestions presented in the report resulted in a significant speedup of approximately 27% on the verification benchmark using cached pre-computation and variable time multi-multiplications. It should be noted that the Ristretto255 implementation leverages

the curve25519-dalek-ng crate, which is forked from curve25519-dalek. The curve25519-dalek-ng crate is currently 214 commits behind the original repository, potentially missing critical security fixes present in the original repo.

In summary, the fastcrypto ECVRF and Ristretto255 implementations have been found to be of high quality, with only minor issues which can be addressed easily.

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, is a deviation from the specification, or can lead to DoS attacks.
Low	Issues harder to exploit (exploitable with low probability), issues that 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

M01: Extra hashing in `ecvrf_encode_to_curve`

Affected Code: fastcrypto/src/vrf.rs (line 99)

Summary: In the `ecvrf_encode_to_curve` implementation, the `expand_message` function applies the hash once before returning the expanded message. Additionally, the `map_to_point` function applies the hash once again before calling the function `from_uniform_bytes`. However, the

Hash-to-Curve RFC Appendix B only hashes once before calling the `from_uniform_bytes` function. This does not lead to security issues but is a deviation from the specification and has negative performance implications.

Suggestion: We recommend using `from_uniform_bytes` directly in the `ecvrf_encode_to_curve` to avoid the additional hashing and potential compatibility issues with other designs.

Suggested Fix: [370694]

Status: Open

M-02: Non-canonical ristretto scalar deserialization

Affected Code: fastcrypto/src/groups/ristretto255.rs (line 191)

Summary: The ristretto scalar deserialization differs from the one upstream as it does not enforce the value to be canonical (i.e., that the input is already reduced). In the context of ECVRF, this is not dangerous as the s and c components of the proof should *not* be assumed unique. Proofs following the spec for nonce derivation are deterministic, but this cannot be checked or enforced by the verifier: an alternative prover can derive k via other means with no clear avenue for detection (i.e they are not unique). In general use, however, it can enable multiple representations for elements that should be unique, making replay attack detection harder.

Suggestion: Consider updating the ristretto scalar deserialization to use the upstream `from_canonical_bytes function`.

Suggested Fix: [ebcd7e1c]

Status: Open

2.3 Low

L-01: Consider removing `verify_output` and bring `verify` inline with the RFC

Affected Code: fastcrypto/src/vrf.rs (line 45)

Summary: The `verify_output` function allows the verification of output given a proof, public key, input, and output. This function is not present in the RFC standard, and its usage could lead to potential pitfalls. The function requires the purported output of the VRF as an input. One can get the output in two ways: fetching it from someone or generating it locally by hashing the proof. The first scenario wastes bandwidth, as the output can be locally generated using the proof. In

the second scenario, we generate the output twice: firstly, to provide it as an input to the function, and, secondly, inside the `verify_output` itself. Removing the `verify_output` function would harden the implementation, as its use case is unclear. Additionally, the current `verify` function returns a boolean, while the RFC specification returns False or (True, Output).

Suggestion: Consider removing the `verify_output` function and modify the `verify` function to align with the RFC specification, returning False or (True, Output). This will help prevent potential issues for downstream developers.

Status: Open

L-02: Use of vartime multiscalar multiplication for faster verification

Affected Code:

- fastcrypto/src/vrf.rs (line 248)
- fastcrypto/src/vrf.rs (line 252)

Summary: Multiscalar multiplication takes the most time during verification. The current implementation uses `RistrettoPoint::multiscalar_mul` to ensure a constant runtime to avoid leaking information. However, the upstream `RistrettoPoint` also allows for `vartime_multiscalar_mul`, which is faster but does not run in constant time. `vartime_multiscalar_mul` can be used for the verification side, where no private information can be leaked.

Suggestion: Consider using `vartime_multiscalar_mul` for the verification side to improve performance without risking the leakage of private information.

Suggested Fix: [53e16c]

Status: Open

2.4 Informational

I-01: Choice of `SUITE_STRING` and `DST`

Affected Code:

- fastcrypto/src/vrf.rs (line 78)
- fastcrypto/src/vrf.rs (line 88)

Summary: Currently, both `SUITE_STRING` and `DST` mention "sui". We understand that the fastcrypto library is primarily used for the Sui blockchain. We recommend using a more neutral identifier to encourage broader adoption of the implemented parameters.

Suggestion: Change the `SUITE_STRING` and `DST` identifiers to a more neutral term, such as "fastcrypto," to promote standardization and wider adoption of the library.
Status: Open

I-02: Consistent use of SHA512 reference type H

Affected Code:

- fastcrypto/src/vrf.rs (line 108)
- fastcrypto/src/vrf.rs (line 137)
- fastcrypto/src/vrf.rs (line 141)

Summary: Some parts of the code use the reference type H for SHA512, while others directly use SHA512. It is recommended to use the reference type H consistently throughout the code.
Suggestion: Modify the code to consistently use the reference type H of SHA512, ensuring uniformity and improved maintainability.
Suggested Fix: [d8edc923]
Status: Open

I-03: Dependency on curve25519-dalek-ng

Summary: The Ristretto255 implementation in the fastcrypto library depends on the curve25519-dalek-ng crate, which is a fork of the curve25519-dalek crate. At the time of the audit, curve25519-dalek-ng is approximately 200 commits behind curve25519-dalek. This discrepancy may result in missed security updates, performance improvements, or other enhancements available in the original curve25519-dalek repository.
Suggestion: We recommend reviewing the changes made in the curve-25519-dalek repository since the fork and consider updating the dependency on curve25519-dalek-ng to include any relevant updates, bug fixes, or security patches. Alternatively, consider switching back to the original curve25519-dalek crate if the changes in curve25519-dalek-ng are no longer necessary or can be incorporated into the fastcrypto library in another way.
Status: Open

I-04: Optimize challenge generation by caching hash state

Affected Code: fastcrypto/src/vrf.rs (line 157)

Summary: The current implementation of `ecvrf_challenge_generation` initializes and updates the hash with the constant `SUITE_STRING` every time it is called. This can be optimized by caching the initial hash state to avoid redundant hashing of the constant `SUITE_STRING`.

Suggestion: Introduce a cached hash state using `Lazy` to store the initial state of the hash with the `SUITE_STRING` already hashed. This would allow reusing the cached state and avoid the unnecessary hashing of `SUITE_STRING` for each invocation of `ecvrf_challenge_generation`.

Suggested Fix: [ae62e9]

Status: Open

I-05: Store compressed state of ECVRFPublicKey to optimize challenge generation

Affected Code: fastcrypto/src/vrf.rs (line 91)

Summary: The current implementation of `ECVRFPublicKey` does not store its compressed state, leading to potential recalculations of the compressed form during challenge generation. This can be optimized by storing the compressed state of the `ECVRFPublicKey` in memory.

Suggestion: Modify the `ECVRFPublicKey` structure to include a field for the compressed representation. Update the serialization and deserialization methods to work with the new structure. This would allow for a more efficient challenge generation process by reusing the stored compressed state instead of recalculating it.

Suggested Fix: [a5bd0a]

Status: Open

I-06: Precompute multiscalar multiplication using VartimeRistrettoPrecomputation

Affected Code: fastcrypto/src/vrf.rs

Summary: Parts of multiscalar multiplication during the verification can be cached to improve the performance of the ECVRF verification process.

Suggestion: Introduce a `challenge_cache` field in the `ECVRFPublicKey` structure and use `VartimeRistrettoPrecomputation` to precompute the multiscalar multiplication between the generator and the public key.

Suggested Fix: [0c0520] (needs more work)

Status: Open

3 Performance Optimizations and Further Benchmarking

In our analysis, we utilized the existing benchmark code for ECVRF, which was developed using the Criterion crate. Our focus was on optimizing the verification function, as it is a crucial component in a blockchain setting where it is performed multiple times by a large validator set.

By generating flamegraphs for the verification benchmark, we were able to identify parts of the code that took the most time. The two multiscalar multiplication operations were the most time-consuming, followed by compressing public keys.

To address these performance bottlenecks, we proposed using vartime multiscalar multiplication (L-02), which is faster than constant-time multiscalar multiplication, and caching public key compression (I-05). Our analysis indicates that implementing both L-02 and I-05 can improve performance by 27%.

We also suggested additional caching techniques in I-04 and I-06. However, further work is required to concretely benchmark the improvements offered by these techniques and determine whether their implementation is warranted. While we strongly recommend adopting L-02 and I-05 due to their demonstrated benefits, more research is needed for I-04 and I-06. A careful trade-off must be considered between code readability and performance gains before deciding to implement these optimizations.

4 Supplementary Information

4.1 Adding Batch Verification Support

The current ECVRF implementation in the fastcrypto library does not support batch verification. Batch verification can improve the efficiency of verifying multiple VRF proofs simultaneously. A recent paper by Badertscher et al. [1], describes a batching technique based on randomized testing that can double verification speed when verifying batches of 1024 ECVRF proofs. The batched proofs do not need to pertain to the same user or the same input. To facilitate this, they change the representation of the proof, replaceing the c component of the VRF proof with U, V as calculated by the prover.

Such proofs use an alternate verification process, where the verifier derives c from the challenge generation function and checks consistency via two verification equations (as opposed to deriving two values U, V). This is effectively a reordering of standard verification: there, we use c, s to derive U, V via the same equations and then use U, V to check

the derived value is equal to the one provided by the verifier. Because standard verification needs to use U, V as hash inputs, its not possible to reduce the effort in the calculation. In alternate verification however, we only need a true/false output out of each equation. In fact, we are only interested in the logical AND of the two. This enables us to reduce effort in exchange of a small soundness error.

We shift the terms so that each equation sums to zero. Due to the SchwartzZippel lemma, a randomized sum of such equations will almost always be non-zero if even one of the terms is non-zero (i.e of one of the equations is false). In computational terms, we only need to perform one (very long) multi-multiplication and some field operations. These optimizations can also be combined with the variant multi-multiplication functions provided by Dalek. The main practical cost to this approach is that U, V are 32 bytes each instead of 16 bytes for c.

Batch-format proofs can be "compressed" to the standard form at negligible computational cost, if not verifying at the same time. Standard proofs can be made batch friendly, though the conversion cost is equivalent to a full verification.

We recommend exploring the addition of batch verification support to the ECVRF implementation in the fastcrypto library, following the approach described in the referenced paper. This could enhance the efficiency and performance of the library, particularly when verifying multiple VRF proofs at once.

4.2 Secret Scalar and Nonce Derivation

The ECVRF standard provides two separate notions of a provers secret: the secret key SK and the (derived) secret scalar x. In general, the secret key is used in two places: first, to produce the public key via x and second to set a prefix used in the nonce generation for proofs. There are two different design paths described in the RFC:

- Suites deriving from P-256 have a simple structure, where the key, the scalar and the prefix are the same value (modulo representation of the secret as a bytestring for the prefix). Suites deriving from P-256 use RFC Section 5.4.2.1 to derive nonces.
- Suites deriving from ed25519 have a more complex structure, where the key is a bytestring hashed into a 64 byte string, half of which is used to derive the secret scalar and half of which is used to derive the nonce as per RFC Section 5.4.2.2.

The current implementation uses the P-256 design for the secret scalar and the ed25519 design for the nonce. This does not directly contradict the guidance in the RFC, as there is no specific dependence between the representation of SK and the nonce derivation. Additionally, the hLen requirement of 64 mentioned in RFC Section 5.4.2.2 is met by the implemented instantiation. We do not believe there exists an issue with this design choice: domain separation is not impacted and the contribution of the secret key to the nonce is similar.

We note that it seems that aligning fully with either of the two RFC designs is not trivial. One either needs to follow the more complex HMAC design from P-256 or follow the scalar derivation from ed25519 which includes operations that are unneeded in ristretto. Part of the ed25519 scalar derivation performs bit operations with the intention of clearing the ed25519 cofactor which does not exist in the ristretto curve. As such, following ed25519 would involve unnecessary operations, otherwise, a degree of divergence may still be present.

References

1. C. Badertscher, P. Gai, I. Querejeta-Azurmendi, and A. Russell. On uc-secure range extension and batch verification for ecvrf. Cryptology ePrint Archive, Paper 2022/1045, 2022. `https://eprint.iacr.org/2022/1045`.
2. B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.
3. H. de Valence, J. Grigg, M. Hamburg, I. Lovecruft, G. Tankersley, and F. Valsorda. The ristretto255 and decaf448 Groups. Internet-Draft draft-irtf-cfrg-ristretto255-decaf448-07, Internet Engineering Task Force, Apr. 2023. Work in Progress.
4. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.
5. S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Velk. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-15, Internet Engineering Task Force, Aug. 2022. Work in Progress.
6. S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 120–130, 1999.

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.