Mysten Fastcrypto ECDSA Secp256k1 Audit

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

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

April 07, 2023
Last update: July 4, 2023

1 Overview

1.1 Introduction

Mysten Labs commissioned Common Prefix to conduct an audit of the ECDSA secp256k1 implementation within their fastcrypto library. The primary objectives of the audit were to assess the security and adherence to relevant standards and investigate performance optimizations and code quality improvements for this specific implementation.

ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographic algorithm utilizing elliptic curves to generate digital signatures. It offers recoverable signatures, which allow for public key recovery from the signature itself. The secp256k1 curve was deterministically constructed to ensure efficient computation while maintaining high security [1, 2].

The fastcrypto implementation of ECDSA secp256k1 primarily serves as a wrapper around the rust-secp256k1 crate, which itself is a Rust wrapper around the C implementation provided by bitcoin-core/secp256k1. The bitcoin-core/secp256k1 library has undergone extensive testing and is battle-tested through its use in the Bitcoin ecosystem [2, 3].

This audit report thoroughly evaluates the ECDSA secp256k1 implementation within the fastcrypto library. The evaluation focuses on aspects such as code security, efficiency, and reliability. Findings from the audit are categorized by severity, and proposed solutions are provided for each identified issue.

1.2 Audited Files

1. [25c36d5c] fastcrypto/src/secp256k1/mod.rs
2. [25c36d5c] fastcrypto/src/secp256k1/recoverable.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 ECDSA Secp256k1 implementation within the fastcrypto library is a well-crafted and secure wrapper around the rust-secp256k1 crate. Only minor engineering-related issues were found during the audit. These issues can be easily fixed by following the recommendations provided in the report.

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

None Found.

2.3 Low

L-01: Inconsistency between `PartialEq`, `PartialOrd`, and `Ord` trait implementations for `Secp256k1PublicKey`

Affected Code:

- fastcrypto/src/secp256k1/mod.rs (line 94)
- fastcrypto/src/secp256k1/mod.rs (line 100)
- fastcrypto/src/secp256k1/mod.rs (line 106)

Summary: The `PartialOrd` and `Ord` trait implementations for `Secp256k1PublicKey` exhibit inconsistency with the `PartialEq` implementation. While the `PartialEq` implementation relies on the underlying `rust_secp256k1::PublicKey` for equality comparison, the `PartialOrd` and `Ord` implementations convert the `rust_secp256k1::PublicKey` to bytes and perform a byte-level comparison. This inconsistency violates the Rust documentation, which explicitly states that `PartialOrd` and `Ord` implementations must be consistent with `PartialEq`.

Suggestion: To avoid unexpected behavior and ensure consistency, it is recommended not to provide implementations for the `PartialOrd` and `Ord` traits within the library. If necessary, applications can implement byte-level comparisons themselves.

Status: Resolved [d1f65d4f8]

2.4 Informational

I-01: Simplify usage of `OnceCell` with `get_or_init` instead of `get_or_try_init`

Affected Code:

- fastcrypto/src/secp256k1/mod.rs (line 155)
- fastcrypto/src/secp256k1/mod.rs (line 220)
- fastcrypto/src/secp256k1/mod.rs (line 274)
- fastcrypto/src/secp256k1/recoverable.rs (line 78)

Summary: The current implementation uses `get_or_try_init` with `OnceCell`, which expects that the initialization function passed to it might return a `Result`. However, in all calls to `get_or_try_init`, an explicit `Ok(something)` is returned. This can be simplified by using `get_or_init` instead and not wrapping the object in `Ok`.

Suggestion: Consider replacing `get_or_try_init` with `get_or_init` in the affected code to simplify the calls and remove the unnecessary wrapping of the object in `Ok`.

Suggested Fix: Here is an example fix

```
--- a/fastcrypto/src/secp256k1/mod.rs
+++ b/fastcrypto/src/secp256k1/mod.rs
@@ -271,8 +271,7 @@ impl Authenticator for
    Secp256k1Signature {
 impl AsRef<[u8]> for Secp256k1Signature {
     fn as_ref(&self) -> &[u8] {
         self.bytes
-            .get_or_try_init::<_, eyre::Report>(||
    Ok(self.sig.serialize_compact()))
-            .expect("OnceCell invariant violated")
+            .get_or_init(|| self.sig.serialize_compact())
     }
 }
```

Status: Resolved [2cc382a83]

I-02: Inconsistency in `PartialEq` trait implementation for `Secp256k1RecoverableSignature`

Affected Code: fastcrypto/src/secp256k1/recoverable.rs (line 95)

Summary: The current implementation of the `PartialEq` trait for `Secp256k1RecoverableSignature` is inconsistent with other structures in the module. While `Secp256k1Signature`, `Secp256k1PublicKey`, and `Secp256k1PrivateKey` utilize the underlying `rust_secp256k1` implementation for `PartialEq`, the `Secp256k1RecoverableSignature` compares byte representations instead.

Suggestion: To ensure consistency, it is recommended to use the `rust_secp256k1` implementation of `PartialEq` for `Secp256k1RecoverableSignature`.

Status: Resolved [2cc382a83]

I-03: Replace hardcoded value with constant in `Secp256k1RecoverableSignature`

Affected Code:

- fastcrypto/src/secp256k1/recoverable.rs (line 62)
- fastcrypto/src/secp256k1/recoverable.rs (line 64)
- fastcrypto/src/secp256k1/recoverable.rs (line 81)
- fastcrypto/src/secp256k1/recoverable.rs (line 82)

Summary: The code utilizes the literal value 64 at multiple locations, but the value is dependent on `SECP256K1_RECOVERABLE_SIGNATURE_SIZE`. Using a constant value `SECP256K1_RECOVERABLE_SIGNATURE_SIZE - 1` instead of the literal 64 ensures consistency in case the `SECP256K1_RECOVERABLE_SIGNATURE_SIZE` is modified in the future.
Suggestion: Replace the literal value 64 with the constant `SECP256K1_RECOVERABLE_SIGNATURE_SIZE - 1` in the affected code to improve code maintainability and ensure consistency.
Status: Resolved [2cc382a83]

3 Supplementary Information

3.1 Testing Signatures with id 2 and 3

We expanded test coverage during the audit to account for high-index (2-3) recoverable signatures. Signatures with such index values only occur when their r value is small enough that it is feasible for either r or $r + q$ to be the x-coordinate of an intermediate curve point, where q is the curve order. Due to the relation between the curve order q and field modulus p, such cases can only occur rarely during normal usage as the fraction $\frac{p-q}{p}$ is negligible. However, it's simple and tractable to construct them on purpose. For this reason, it is important that they be treated consistently by both branches of the implementation (i.e., converting between recoverable and non-recoverable forms of the same signature whilst keeping track of the related public key should never change the output of the verification function). We have added testing to confirm this is indeed the case.
Pull Request: github.com/MystenLabs/fastcrypto/pull/607

References

1. Certicom research, sec 2: Recommended elliptic curve domain parameters, 2010. https://www.secg.org/sec2-v2.pdf.
2. Bitcoin wiki secp256k1, 2019. https://en.bitcoin.it/wiki/Secp256k1.
3. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

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.