OpenZeppelin | security

ZKsync Crypto Precompile Audit

April 3, 2025

Table of Contents

Summary

Type	Precompile
Timeline	From 2025-02-25 To 2025-03-12
Languages	Rust

Total Issues	9 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	5 (5 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	2 (2 resolved)
Notes & Additional Information	1 (0 resolved)

Scope

We audited pull request #77 of the matter-labs/zksync-crypto repository at commit ccf5fa3.

In scope were the following files:

```
crates
└── boojum
    └── src
        └── gadgets
            ├── curves
            │   ├── sw_projective
            │   │   └── mod.rs
            │   ├── zeroable_affine
            │   │   └── mod.rs
            ├── non_native_field
            │   ├── implementations
            │   │   ├── impl_traits.rs
            │   │   ├── implementation_u16.rs
            │   │   └── mod.rs
            │   └── traits
            │       └── mod.rs
            ├── tower_extension
            │   ├── algebraic_torus.rs
            │   ├── fq12.rs
            │   ├── fq2.rs
            │   ├── fq6.rs
            │   ├── mod.rs
            │   └── params
            │       ├── bn256.rs
            │       └── mod.rs
            ├── traits
            │   ├── hardexp_compatible.rs
            │   └── mod.rs
            ├── u256
            │   └── mod.rs
            └── u512
                └── mod.rs
└── pairing
    └── src
        └── bn256
            ├── ec.rs
            └── mod.rs
            └── pairing_certificate.rs
```

System Overview

The code under review is part of the `pairing` and `boojum` crates within Matter Labs' `zksync-crypto` repository, a cryptographic library that supports ZKsync—a layer-2 scaling solution for Ethereum. These crates implement elliptic curve pairings using the BN256 curve, also known as alt_bn128 or BN254 in different contexts. They enable zk-SNARKs, a zero-knowledge proof critical for ZKsync, enhancing Ethereum transaction scaling securely and efficiently.

The `pairing` crate, located in `crates/pairing/src/bn256`, handles BN256 curve arithmetic through files like `ec.rs` and `mod.rs`. It computes pairings natively on the host system, supporting ZKsync's rollup proofs and signature verification outside arithmetic circuits. The audited code in the `pairing` crate employs precomputed line functions for the Miller loop, a key step in pairing computation, to enhance performance for zk-SNARK verification. This approach improves efficiency by reducing the computational cost of pairing operations.

The `boojum` crate is located in `crates/boojum/src`. It focuses on circuit-based BN256 operations, including affine and projective curve arithmetic in `gadgets/curves`, tower field extensions in files under `gadgets/tower_extension` and `algebraic_torus.rs`, and non-native field arithmetic in `gadgets/non_native_field`. This crate encodes pairing computations as arithmetic circuits, enabling ZKsync's prover to generate zk-SNARK proofs. The audited enhancements optimize these circuits with techniques like algebraic torus arithmetic, reducing the complexity and cost of proof generation.

Security Model and Privileged Roles

This audit targeted the files listed above (in `crates/pairing/src/bn256` and `crates/boojum/src`), which have dependencies that are outside the scope of this audit. While we reviewed the dependencies, unchanged parts outside this scope cannot be reliably deemed to be correct.

The `pairing` crate assumes a secure native environment, trusting precomputed Miller loop functions for rollup proofs and signatures. The `boojum` crate uses arithmetic circuits, assuming a trusted prover-verifier setup where the prover accesses witness data but ensures verifiable zero-knowledge proofs. Both crates trust the hardness of pairings on BN256 curves, with no additional roles beyond standard cryptographic assumptions (e.g., honest computation).

However, with the advancement of tower field sieve techniques, the effective security of BN256 (due to these advances) is reduced to approximately 100 bits. Thus, although unbroken in practice, BN254 is unsuitable for future use due to this weakness.

High Severity

H-01 Incorrect Implementation of `pow_u32` Exponentiation for Even Exponent

In `algebraic_torus.rs`, the `pow_u32<CS, S: AsRef<[u64]>>(&mut self, cs: &mut CS, exponent: S)` function, when $S = 2^k$, outputs $\frac{\gamma}{g}$ incorrectly for all k as a result of starting with `Self::zero` and a static `base`. This error persists in all the cases where S is even, breaking the exponentiation in those instances. If used in pairing computations, this could lead to incorrect pairing evaluations, potentially allowing false zk-proofs.

Consider precomputing using the first bit, and then starting from the second bit, updating `base` in each iteration.

Update: *Resolved in PR #87 at commit 65c890d.*

H-02 Missing Subgroup Check for G2 Points

In `pairing_certificate.rs`, points in G2 are used in pairing operations without explicit verification that they belong to the correct prime-order subgroup. If a point (x, y) does not lie in this subgroup, the pairing computation may yield incorrect results, potentially compromising the validity of proofs.

Consider implementing a subgroup validation check by ensuring that $[r]P = O$, where (r) is the prime order of the subgroup and (O) is the identity element. A faster way to enforce this is to check the conditions of Proposition 3 in this paper.

Update: *Resolved. This is not an issue. The Matter Labs team clarified that subgroup checks are enforced on the circuit level and since this code is only used for witness generation, the witness generation process itself does not enforce subgroup checks.*

H-03 Unimplemented `allocate_constant` Causes Panic in `fq2.rs`

The function `allocate_constant` in the `CSAllocatable` trait is unimplemented.

This results in a panic when `allocate_constant` is called to allocate constants for `fq2`, and the proof generation won't complete.

Consider overriding `allocate_constant` in `fq2.rs` by wrapping inner constant allocations appropriately, or implementing this function.

Update: *Resolved in pull request #88 at commit 09cbca4.*

H-04 Lack of Zero Check in Inverse Computation for Tower Extensions

The inverse functions in fq2.rs, fq6.rs, and fq12.rs do not currently verify whether the element is zero before computing the inverse.

This may lead to undefined behaviour when attempting to invert a zero element.

Consider adding a check for a zero element in the respective field and include error management to return an error when the element is zero.

Update: *Resolved in pull request #89 at commit 15b2ca0.*

H-05 Incorrect Computation of NEGATIVE_ONE Constant in fq.rs

In the `fq.rs` file, the constant `NEGATIVE_ONE`, which is supposed to represent $(-((2^{256}) \mod q) \mod q)$, is incorrectly computed.

This constant is used in the field extension definitions and may cause errors in field extension and arithmetic operations involving it, leading to incorrect results.

Consider correcting `NEGATIVE_ONE` to its correct value: `['0x68c3488912edefaa', '0x8d087f6872aabf4f', '0x51e1a24709081231', '0x2259d6b14729c0fa']`.

Update: *Resolved in pull request #90 at commit 729cf0a.*

Medium Severity

M-01 Converting a potentially 512-bit number to a 256-bit number, causing panic

In `crates/boojum/src/gadgets/u256/mod.rs`, line 391 has the code:

```
let q: U256 = q.try_into().unwrap();
```

Here, `q` equals $\lfloor \frac{ab}{m} \rfloor$, where a and b are 256-bit numbers. Since `q` can be up to 512 bits, the conversion to `U256` may fail. If it does, `unwrap()` causes the `modmul` function to panic, disrupting the proof flow.

Consider using `U512` for `q` or adding error handling to avoid panics.

Update: *Resolved in pull request #92.*

Low Severity

L-01 Silent Debug Assertions `utils.rs` Under Release Mode

In the file `utils.rs`, line 8, the function contains several debug assertions (e.g., `debug_assert!`). These assertions are disabled when the code is compiled in release mode, a standard Rust behaviour. As a result, conditions checked by these assertions are not verified during production execution, potentially allowing errors to go unnoticed in the `boojum` crate.

In a cryptographic proofs, where arithmetic correctness is critical for proof generation and verification, silent failures could lead to invalid proofs or performance issues.

Consider replacing `debug_assert!` with `assert!` in release mode.

Update: *Resolved in pull request #94.*

L-02 Redundant Computation in Line Coefficients and Point Operations

In the current implementation of line function computations in `pairing_certificate.rs`, the slope `alpha` is computed separately in the `line_double` and `line_add` functions. Simultaneously, point operations like `t.double()` and `t.add_assign_mixed()` recompute `alpha`, rather than reusing the value.

This results in redundant field arithmetic operations, reducing the computational efficiency of pairings and proof verification.

Consider integrating the computation of `alpha` and `mu` directly into the point operations to minimize repeated calculations and enable the reuse of intermediate results.

Update: *Resolved in pull request #93.*

Notes & Additional Information

N-01 Incorrect Comments and Typos in Codebase

Throughout the codebase, we identified the following typos and incorrect comments that should be addressed:

1. In `fq.rs`, line 103 incorrectly states `Fq2(u + 1)**(((2q^0) - 2) / 3)`. The correct expression should be `Fq2(u + 9)**(((2q^0) - 2) / 3)`. This correction also applies to the following lines: **103, 108, 113, 118, 123, 128, 137, 142, 147, 152, 157, 162, 167, 172, 177, 182, 187, and 192.**
2. In `sw_projective/mod.rs`, the comment at line 213 should be corrected to: `// y3 = y3 + t0`
3. In `ec.rs`, line 592 should be updated to: `unimplemented!("on curve check is not implemented for BN-256 projective")`

4. In `pairing_certificate.rs`, to clarify the comment at line 118, it should be changed to: `// and compute c0 = 1, c3 = - lambda * p.x / p.y = lambda * x', c4 = - mu / p.y = mu * y'`

Consider applying these corrections to help improve code readability and prevent future confusion, especially for the developers working on cryptographic functions and the BN-256 curve.

Update: *Acknowledged, will resolve. The Matter Labs team stated that they would like to postpone the fix and incorporate it into the upcoming improvements.*

Recommendations

Optimization of the Function `decompress` in `algebraic_torus.rs`

In file algebraic_torus.rs, the function `decompress` can be optimized based on the following observation:

Given that

$$\frac{g+w}{g-w} = \frac{g^2+(u+9)}{g^2-(u+9)} + \frac{2g}{g^2-(u+9)}w,$$

and noting that $d = u + 9$ is a constant that is already precomputed, the expression can be further simplified to

$$1 + 2D^{-1}(d + gw),$$

where, $D = g^2 - d$ is the denominator.

This approach reduces constraints and computational overhead by leveraging the special structure of the expression, minimizing multiplications and redundant operations. As a result, it improves the efficiency of pairing computations and proof verification.

Implementation Steps

1. Import the precomputed constant $d = u + 9$ as a constant in $\mathbb{F}_{p^2}$
2. Compute $D = g^2 - d$ (one squaring and one subtraction in $\mathbb{F}_{p^6}$.
3. Compute D^{-1} (one inversion in $\mathbb{F}_{p^6}$).

4. Compute $2D^{-1}$ (one doubling in $\mathbb{F}_{p^6}$).
5. Add 1 in $\mathbb{F}_{p^2}$.

Conclusion

The audited code provides a cryptographic library for native pairing-based signature generation, constrained cryptographic operations using non-native field arithmetic, and circuits supporting field extensions.

During the audit, several high-severity issues were identified. In addition, the audit identified few improvement areas, particularly in testing, error handling, and documentation. A more thorough unit testing suite would help ensure that all operations are sound, edge cases are properly handled, and any silent or unimplemented errors are identified. While some errors, such as division by zero, are implicitly managed by the circuit, explicitly documenting these cases would improve clarity and robustness. Additional documentation could also clarify the rationale behind certain design choices, particularly the complex overflow tracking and reduction/normalization scheme used to simulate non-native field arithmetic in-circuit. While not immediately threatening to system security, addressing these aspects would enhance the reliability and maintainability of the protocol.

Despite these findings, communication with the team was notably fast and friendly and we appreciate the collaboration with the Matter Labs team on this project.