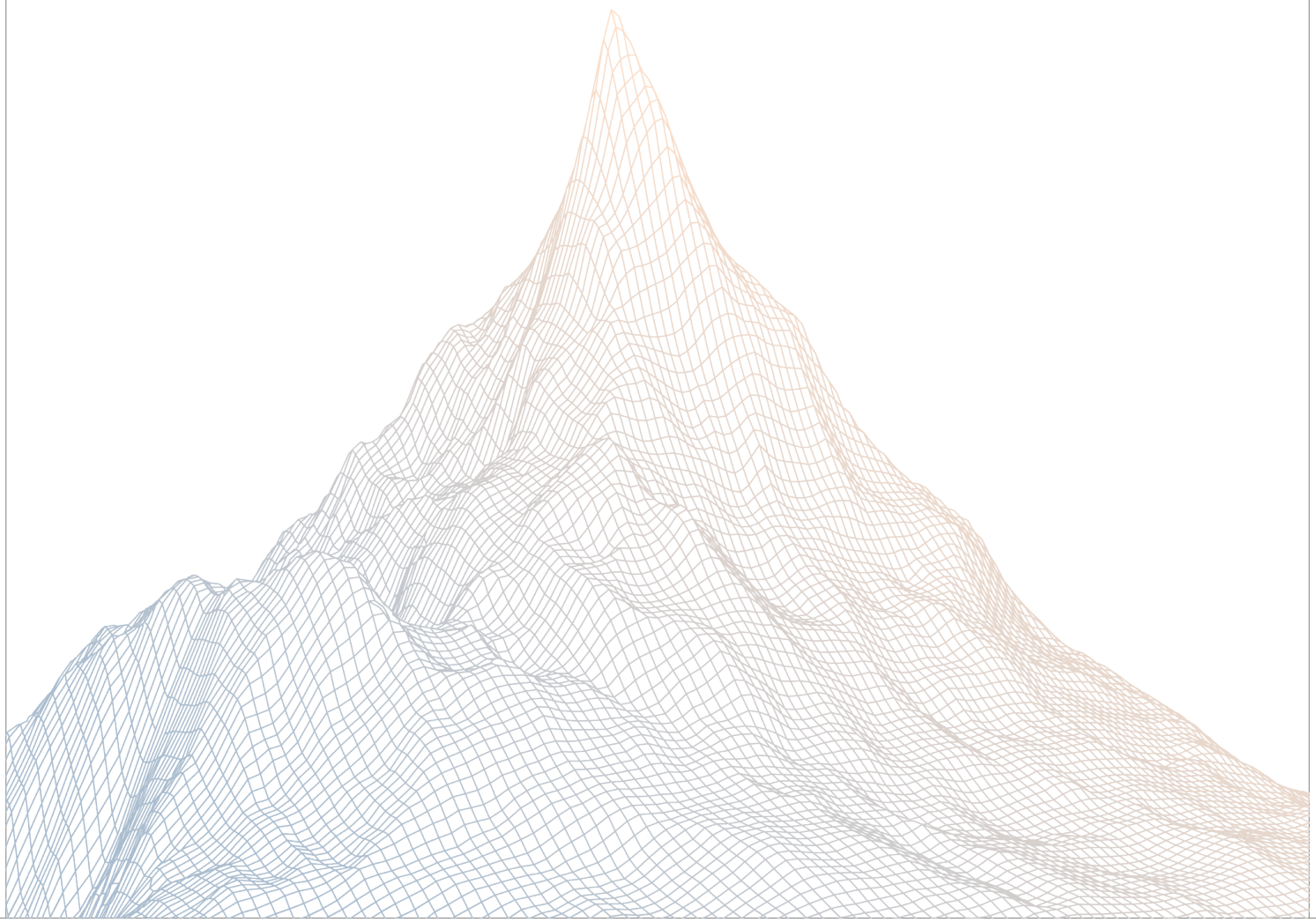
Zenith

Alpen Labs

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: November 17th to December 22nd, 2025

AUDITED BY: Manish Kumar
Alex Liao

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About Alpen 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 6

4.1 High Risk 7

4.2 Medium Risk 11

4.3 Low Risk 13

4.4 Informational 18

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Alpen

Alpen gives developers the freedom to program nearly any locking conditions for BTC imaginable, limited only by the Alpen block size and gas limits. This enables developers to create new kinds of applications for BTC with features such as: New signature types, vaults, subscriptions and strong privacy

2.2 Scope

The engagement involved a review of the following targets:

Target	g16
Repository	https://github.com/alpenlabs/g16
Commit Hash	`7d7e002ac9a78700e475750dd45a3672edf394f7`

Files	`bn254/g1.rs`
	`bn254/g2.rs`
	`bn254/pairing.rs`
	`bn254/final_exponentiation.rs`
	`groth16.rs`

Target	ckt
Repository	https://github.com/alpenlabs/ckt
Commit Hash	`1b64d000991e660ae080dd423244fcc6f73fdde0`
Files	`crates/fmtv5-types/*`

2.3 Audit Timeline

November 17, 2025	Audit start
December 22, 2025	Audit end
February 3, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	3
Medium Risk	2
Low Risk	5
Informational	4
Total Issues	**14**

3

Findings Summary

ID	Description	Status
H-1	Missing group validation for the input points of Groth16 circuit	Resolved
H-2	Incorrect random number generation in Testing	Resolved
H-3	Missing subgroup check for G1 and G2 points during decompression	Resolved
M-1	Incorrect checks for primary input wires	Resolved
M-2	Use checked_add to calculate write offset in flush_io_buffer	Resolved
L-1	Validate if num_outputs < total_gates in the header as required in the spec	Resolved
L-2	Use checked_add instead of saturated_add or non-checked add for calculating total gates in the header	Resolved
L-3	Unwanted panic while normalizing points to affine	Acknowledged
L-4	Missing assertions for equal points in add_montgomery	Resolved
L-5	Calculating inverse of Z in projective_to_affine_montgomery may cause panic	Resolved
I-1	Potential Alignment Violation in Block as per spec	Resolved
I-2	Spec of checksum verification should include padding	Resolved
I-3	Manually close the file in CircuitWriter after finalizing	Resolved
I-4	Same struct type used for both projective and affine representation	Acknowledged

4

Findings

4.1 High Risk

A total of 3 high risk findings were identified.

[H-1] Missing group validation for the input points of Groth16 circuit

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target:

- groth16.rs

Description:

In `groth16_verify` and `groth16_verify_compressed`, there's no validations on whether the input points are in the correct prime-order subgroup. Coupling with #1 that is used in `groth16_verify_compressed` , invalid points can be used in the Groth16 verification that may break the paring computation and allow attackers to forge a Groth16 proof.

Recommendations:

Similar to #1, adds subgroup check for the input G1 and G2 points.

Alpen: Resolved with PR-42

Zenith: Verified.

[H-2] Incorrect random number generation in Testing

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target:

- src/gadgets/bn254/g1.rs
- src/gadgets/bn254/g2.rs

Description:

The fn `rnd_g2` and `rnd_g1` is used to generate random g1 and g2 points to test multiple functions in the codebase.

```
pub fn rnd_g1(rng: &mut impl Rng) -> ark_bn254::G1Projective {
        ark_bn254::G1Projective::default() * rnd_fr(rng)
    }
```

Here, the G1 point is taken to be default (same for G2) which result into rnd_g1 always giving zero points for G1 and G2 instead of random points. Changing the `ark_bn254::G1Projective::default()` to `ark_bn254::G2Projective::generator()` (similarly for G2) to generate random g1 and g2 points results into failure of multiple tests. Testing with always a default value instead of random g1 and g2 points result into extremely limited test coverage and bugs/logic issues can go completely undetected.

Recommendations:

Change the `rnd_g2` and `rnd_g1` to generate random numbers:

- for G2:

```
pub fn rnd_g2(rng: &mut impl Rng) -> ark_bn254::G2Projective {
        ark_bn254::G2Projective::generator() * rnd_fr(rng)
    }
```

- similarly for G1

```
pub fn rnd_g1(rng: &mut impl Rng) -> ark_bn254::G1Projective {
        ark_bn254::G1Projective::generator() * rnd_fr(rng)
    }
```

and then fix all the failing tests

Alpen: The random number generation issue is resolved with PR-59

Zenith: Verified.

[H-3] Missing subgroup check for G1 and G2 points during decompression

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target:

- groth16.rs

Description:

When performing decompression, there are not validations on whether the resulting point is in the correct prime-order subgroup. In the case of `decompress_g1_from_compressed`, since `Fq::sqrt_montgomery` does not verify whether a point is a quadratic residue, the result can be an invalid point of the curve. As for `decompress_g2_from_compressed`, the resulting point can be on the curve but not in the correct prime-order subgroup. Thus explicit subgroup check is necessary for G2 points. Using points not in the correct prime-order subgroup may break the paring computation and allow attackers to forge a Groth16 proof.

Recommendations:

For G1 decompression, verify the result of `Fq::sqrt_montgomery` to make sure it is a valid elliptic curve point. For G2 decompression, add explicit subgroup check for the resulting point. A simple way of subgroup check for point **P** is to see if **[r]P = O**, where r is the order of G2 and O is the identity. A more efficient way can be found in [Section 4.3, HGP].

Alpen: Resolved with PR-42. G1 group check commit and G2 group check commit

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Incorrect checks for primary input wires

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target:

- lvl/prealloc.rs

Description:

In lvl/prealloc.rs, there's a check `wire ⩽ primary_inputs + 2` for whether the wire is a primary input. However wire `primary_input + 2` should not be a primary input since there are in total `primary_inputs + 2` primary inputs and the index starts form 0. That particular output wire will be treated as a primary input and return its `wire_id` as the `slab_id`. It will also never be deallocated since the credits check part will never be run. In the current implementation the first output wire will always have the same slab_id, so it runs without any issue. However if there are changes to the slab allocator logic the current implementation might fail.

Recommendations:

As described in the description.

Alpen: Resolved with PR-55

Zenith: Verified

[M-2] Use checked_add to calculate write offset in flush_io_buffer

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target:

- v5/a/writer.rs

Description:

In `flush_io_buffer()`, next offset is calculated by unchecked add. If the value overflow, the buffer will be written in an incorrect position, leading to a corrupted v5a file.

Recommendations:

Use `checked_add` similar to `flush_io_buffer` in v5/c/writer.rs.

Alpen: Resolved by updating the calculation of `next_offset` to `checked_add` in PR-62.

Zenith: Verified.

4.3 Low Risk

A total of 5 low risk findings were identified.

[L-1] Validate if num_outputs < total_gates in the header as required in the spec

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target:

- v5/c/header.rs

Description:

As required in the spec of v5c, num_outputs should be smaller than total_gates in the header. Currently there's no check against this requirement.

Recommendations:

Add checks for `num_outputs < total_gates` in the validate function of headers.

Alpen: The check for the num_outputs of v5c is updated to num_outputs < total_gates + num_inputs in both header.rs and the spec with PR-66

Zenith: Verified.

[L-2] Use checked_add instead of saturated_add or non-checked add for calculating total gates in the header

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target:

- v5/a/mod.rs
- v5/c/header.rs

Description:

In v5a, `total_gates` is calculated using unchecked add. Similarly in v5c, `total_gates` is calculated using `saturating_add`. Both methods can return incorrect gate count if `xor_gates + and_gates` overflow in u64. The method is used during the calculation of checksum in the reader and thus can cause the checksum verification to fail.

Recommendations:

Use `checked_add` to check if the gate count overflow.

Alpen: Resolved with the calculation of total gates being updated to checked add in PR-63

Zenith: Verified.

[L-3] Unwanted panic while normalizing points to affine

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target:

- src/gadgets/bn254/pairing.rs
- src/gadgets/bn254/pairing.rs

Description:

The functions `g1_normalize_to_affine` and `g2_normalize_to_affine` normalizes the projective points to affine. This method however does not take into account the point at infinity and will cause unwanted panic while trying to convert a point at infinity projective point to affine since the point and infinity has z component 0 and `Fq::inverse_montgomery` (or Fq2::inverse_montgomery) will panic for 0. Same issue has been raised earlier here: https://github.com/zenith-security/2025-11-alpen-labs/issues/1. It should be noted that the arkworks handles the point and infinity conversion i.e., converting a infinity point from projective coordinates to affine does not panic due to their `Affine` representation which can also represent points at infinity. Since the circuit implementation intended to follow the implementation compatible with arkworks, the edge cases should be compatible as well(in this case the point at infinity).

Recommendations:

Add explicit zero check in those functions to handle points at infinity as well so that unwanted panic can be avoided.

Alpen: Acknowledging that it won't be fixed due to being part of unused code.

[L-4] Missing assertions for equal points in `add_montgomery`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target:

- src/gadgets/bn254/g1.rs
- src/gadgets/bn254/g2.rs

Description:

The `add_montgomery` function uses point addition method using Jacobian Projective Coordinates as mentioned in the ref. It must be noted that the method is applicable only for points P, Q such that $P, Q \neq \infty$ and $Q \neq P$ as mentioned in the pdf. The implementation however only takes into the account $P, Q \neq \infty$ by checking if the z-coordinate of either input is 0 and uses a multiplexer to return the other point but fails to have a check that $Q \neq \pm P$. The $Q \neq -P$ holds implicitly but $Q \neq P$ does not (point doubling done separately). Right now in the codebase it is used for the scalar multiplication and msm so this can be a deliberate optimization but if the function is exposed as a general purpose fn to construct circuits, this can be an issue and can be exploited.

Recommendations:

If the gadgets is strictly for internal components then checks can be avoided as the algorithm naturally avoids $P = Q$ but if the function is exposed as public where p and q are directly provided by user then circuit assertions should be added to ensure that $P \neq Q$.

Alpen: Resolved with PR-63. Since the circuit is to be used only for groth16, the scope of the function was limited only within the crate and a comment was added stating that the fucntion doesn't work for `P=+-Q`

Zenith: Verified.

[L-5] Calculating inverse of Z in projective_to_affine_montgomery may cause panic

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target:

- groth16.rs

Description:

In `projective_to_affine_montgomery`, there is no zero check on point Z. If a user provides an incorrect proof, it might cause the Groth16 verification to panic instead of returning the expected result.

Recommendations:

Add explicit zero check in `projective_to_affine_montgomery`

Alpen: The zero check is added in PR-37.

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] Potential Alignment Violation in `Block` as per spec

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target:

- crates/fmtv5-types/src/v5/c/block.rs

Description:

The spec in `crates/fmtv5-types/src/v5/c/SPEC.md` describes 8-byte alignment for optimal memory access but `GateV5c` has 4 bytes alignment and consequently `Block` also has 4 byte alignment. So if a raw pointer *const u8 aligned to 4 but not 8 is casted into *const Block may result into undefined behaviour or may not be efficient.

Recommendations:

Add `#[repr(C, align(8))]` to Block to make it 8 byte aligned as per the mentioned spec.

Alpen: The attribute for the 8-byte alignment is added in PR-55

Zenith: Verified.

[I-2] Spec of checksum verification should include padding

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target:

- v5/c/SPEC.md

Description:

In the spec of v5c, checksum verification will exclude the padding for gate blocks and outputs. However the implementation includes both paddings in the checksum.

Recommendations:

Update the spec of checksum verification to match the implementation.

Alpen: Resolved by having the specification updated to include padding with PR-65

Zenith: Verified

[I-3] Manually close the file in CircuitWriter after finalizing

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target:

- a/writer.rs
- c/writer.rs

Description:

In the documents of monoio::fs::File, it is recommended to manually call the `close()` function in order to guarantee the file is closed successfully.

Recommendations:

If the `CircuitWriter` is designed to not be used after calling `finalize()`, consider adding `self.file.close().await?;` after `self.file.sync_all().await?;`.

Alpen: Resolved by having the files in v5a and v5c explicitly closed in PR-64

Zenith: Verified.

[I-4] Same struct type used for both projective and affine representation

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target:

- src/gadgets/bn254/pairing.rs

Description:

In multiple functions inside the file pairing.rs, the affine points are also represented using the Projective struct types (G1Projective & G2Projective) with the condition that z component is equal to 1. While this is not an issue, this creates confusion across the codebase and is prone to logic errors if not handled carefully. Therefore, it is always recommended to have different type for different representations(affine and projective).

Recommendations:

Define a separate type for handling affine representations.

Alpen: Won't be addressed due to low impact and high engineering risk to fix.

Zenith: Acknowledged.