CANTINA

Aztec stdlib field

Security Review

Cantina Managed review by:

Ed255, Lead Security Researcher
Jakub Heba, Associate Security Researcher

December 16, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Aztec Labs was founded in 2017, and has a team of +50 leading zero-knowledge cryptographers, engineers, and business experts. Aztec Labs is developing its namesake products: AZTEC (a privacy-first L2 on Ethereum) and NOIR (the universal ZK language).

From Jul 8th to Jul 22nd the Cantina team conducted a review of aztec-packages on commit hash 458fb330. The team identified a total of **17** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	3	3	0
Gas Optimizations	0	0	0
Informational	14	14	0
Total	**17**	**17**	**0**

3 Findings

3.1 Low Risk

3.1.1 No restriction on `num_bits` in `ranged_less_than`

Severity: Low Risk

Context: field.hpp#L420

Description: Using `ranged_less_than` with a value of `num_bits` greater or equal than the number of bits of the `native` field makes the constraints unsound: the range check will just accept everything. I'm not sure if the creation of the range constraint with such big value for `num_bits` will compile and/or run without any crash/exception.

Recommendation: Add a static assert to check that `num_bits` is lower than the number of bits minus one of the native field (`K=2^num_bits` should always be less than half of the modulus of the native field).

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.1.2 Missing builder context validation in field operations

Severity: Low Risk

Context: field.cpp#L121

Description: In arithmetic operations, the code selects a working context using patterns like `ctx = (context == nullptr) ? other.context : context` in operator+/operator*, `ctx = (context) ? context : other.context` in `divide_no_zero_check`, and `ctx = first_non_null(context, other.context, third.context)` in methods like `madd` and `add_two`.

When operands have different non-null contexts, these operations proceed using one Builder's context while reading witness indices that belong to different Builders. Since witness indices are positions in a specific Builder's variables array, using indices from `builder2` in `builder1`s context creates gates with wrong variable references.

For example, if `builder1` and `builder2` both have a witness at index 0 with values 10 and 20 respectively, an operation between them would incorrectly use value 10 twice instead of 10 and 20.

Recommendation: Add validation to detect context mismatches when both operands are non-constant.

```
if (!is_constant() && !other.is_constant()) {
    ASSERT(context == other.context);
}
```

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.1.3 The `set_public` on constant `field_t` passes `IS_CONSTANT` as a wire index

Severity: Low Risk

Context: field.hpp#L348

Description: The `set_public` is doing:

```
return context->set_public_input(normalize().witness_index);
```

but for a pure constant where `witness_index == IS_CONSTANT`, it passes `IS_CONSTANT` to `set_public_input`, which isn't a valid wire index. In practice, that will either crash or silently misuse a public input slot.

Recommendation: We recommend adding an `ASSERT(!is_constant(), "error")` to mitigate the issue.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 Assumption over ranged_less_than is overly restrictive, method comment has some mistakes

Severity: Informational

Context: field.hpp#L414-L443

Description: The documentation of the method `ranged_less_than` mentions that `a` and `b` are assumed to be `< 2^{num_bits} - 1` but this check is not practical and over restrictive.

It's practical to require an input to be of a certain number of bits. This can be achieved with a range check. The assumption that a number takes a number of bits would be `a <= 2^{num_bits} - 1`. Notice the lower equal comparison used here VS the lower than comparison found in the documentation.

The constraints of this method also work when `both a and b are <= 2^{num_bits} - 1`.

The internal method comment is also a slightly incorrect with respect to the implementation.

- The implementation uses `K = 2^{num_bits}` but the comment uses `K = 2^{num_bits} - 1`.
- If `q == 1` then `0 < b - a - 1` is not true. Given `a = 0, b = 1` we have `0 < 1 - 0 - 1` $\Leftrightarrow$ `0 < 0` which is false.

Recommendation: Change the method documentation to say.

```
This method *assumes* that both a and b are <= 2^{num_bits} - 1
```

or

```
This method *assumes* that both a and b are < 2^{num_bits}
```

Change the method comment to something like this:

```
// Let q = (a < b)
// Assume both a and b are < K where K = 2^{num_bits}
//    q == 1 <=>  0 < b - a     < K =>  0 <= b - a - 1     < K
//    q == 0 <=>  0 < b - a + K < K =>  0 <= b - a + K - 1 < K
// i.e. for any bool value of q:
//    (b - a - 1) * q + (b - a + K - 1) * (1 - q) = r < K
//     q * (b - a - b + a) + b - a + K - 1 - (K - 1) * q - q = r < K
//     b - a + (K - 1) - K * q = r < K
```

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.2 Use `get_value` to get a boolean value

Severity: Informational

Context: field.cpp#L46

Description: When creating a `field_t` from a `bool_t` in the constant case, using the `bool_t->get_value` method makes the code clearer than manually computing the boolean value with `other.witness_bool ^ other.witness_inverted`. If there's a future change in the `bool_t` internal implementation this would still work.

Recommendation:

```
additive_constant = other.get_value() ? bb::fr::one() : bb::fr::zero();
```

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.3 Bool_t has a constructor that takes context and value

Severity: Informational

Context: field.cpp#L79-L80

Description: This code can be simplified with an already existing constructor.

Recommendation:

```
bool_t<Builder> result(context, additive_constant == bb::fr::one());
```

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.4 Inconsistent naming of plonk gate terms

Severity: Informational

Context: field.cpp#L529-L533

Description: The plonk gate terms are written in different ways along the code-base. I've encountered the following three styles:

```
q_m * w_l * w_r + q_l * w_l + q_r * w_r + q_o * w_o + q_4 * w_4 + q_c
```

```
q_m * a * b + q_1 * a + q_2 * b + q_3 * c + q_c
```

```
mul_scaling * a * b + a_scaling * a + b_scaling * b + c_scaling * c + d_scaling * d + const_scaling
```

Recommendation: Use the same style/terms around the code-base where possible. Being consistent greatly helps the reader of the code to understand the logic with less overhead.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.5 `madd` could be optimized for the case where 2 inputs are constant

Severity: Informational

Context: field.cpp#L509

Description: The current implementation of `madd` optimizes the case where the 3 inputs are constants, otherwise it generates one constraint. If two inputs are constant (and the third one contains a witness) the constraint can be skipped. The same optimization could be applied to `add_two`.

Recommendation: If there are chances of using `madd` with two constant inputs, applying the described optimization would reduce the total number of gates in a circuit. This could be evaluated by taking different existing circuits and counting how many instances of `madd` are being called with two constant inputs.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.6 Unnecessary witness multiplied by 0 in `normalize`

Severity: Informational

Context: field.cpp#L652

Description: The `normalize` method creates a gate that does `this.mul * this.v + 0 * this.v + result.v * (-1) + this.add = 0`. The assignment of the term `b = this.v` stands out because the multiplicative term is 0.

Recommendation: Assigning `context->zero_idx` to the `b` term would make the code more clear as the reader doesn't have to figure out why the `witness_index` is used twice in the gate, only to realize that in one instance it's being multiplied by 0. For example the method `assert_is_zero` follows this pattern of using `context->zero_idx` and is more readable.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.7 `assert_equal` optimization

Severity: Informational

Context: field.cpp#L930-L932

Description: When `lhs` and `rhs` in `assert_equal` are non-constant, both `field_t`s are normalized. If both `field_t`s weren't already normalized, two gates will be used. For such a case, instead of doing a copy constraint, a single gate could be used that constraints the two `field_t`s to be equal.

Recommendation: When `lhs` and `rhs` are not normalized and not constant, instead of normalization and copy constraining apply the following gate:

```
lhs - rhs = 0
lhs.v * lhs.mul + lhs.add - (rhs.v * rhs.mul + rhs.add) = 0
lhs.mul * lhs.v + (-rhs.mul) * rhs.v + (lhs.add - rhs.add) = 0
---
a = lhs.v
b = rhs.v
q_m = 0
q_l = lhs.mul
q_r = -rhs.mul
q_c = lhs.add - rhs.add
```

This reduces the number of used gates from 2 to 1 in this case.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.8 Confusing comment

Severity: Informational

Context: field.cpp#L1149-L1150

Description: In `accumulate`, the calculation of the witness result has a comment about a conditional addition of the constant values when `input` contains non-constant `field_t` elements. But the code earlier has checked for the case when there are 0 non-constant `field_t` elements in the `input` with an early return:

```
if (accumulator.empty()) {
    return constant_term;
}
```

So at this point of the function, we know that the input contains non-constant `field_t` elements. And we unconditionally calculate the witness of the addition over all inputs.

Recommendation: Update the comment to reflect the current code.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.9 Incorrect documentation of `slice` method

Severity: Informational

Context: field.cpp#L1243-L1244

Description: The slice method slices a field into 3 parts which are named `lo`, `slice`, `hi` in the code corresponding to the least significant part, the middle part and the most significant part of the original element. The code has some comments that treat the bit `0` as the least significant and the bit `255` as the most significant. The code describes the following bit ranges for each part: `lo: [0,lsb - 1]`, `slice: [lsb, msb]` and `hi: [msb + 1, 255]`.

Nevertheless the method documentation describes the bit ranges as `[0, msb-1]`, `[msb, lsb]`, and `[lsb+1, 256]`, which is inconsistent with the code comments. It's also inconsistent with the order of the returned elements.

Recommendation: Update the method documentation to describe the bit ranges as `[0,lsb - 1], [lsb, msb] and [msb + 1, 255]`.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.10 `decompose_into_bits` with `num_bits = 253` does unnecessary aliasing check

Severity: Informational

Context: field.cpp#L1376

Description: The `decompose_into_bits` method performs an aliasing check to make sure the bit decomposition has a unique solution (the canonical representation). The extra constraints for that are only supposed to be generated when an aliasing can occur. Nevertheless when `num_bits = 253` the aliasing check constraints are generated but there can't be aliasing:

The code calculates `num_bits_modulus_minus_one = 253`. This is correct because the log2 of the modulus is 253.6: so it takes 254 bits to represent an `fr` element. If `num_bits = 253` the representation must be unique, because an overflow would require an extra bit (with 254 bits we can represent 2^{254} values which is bigger than $2^{253.6}$).

Recommendation: Only apply the aliasing check constraints when `num_bits > num_bits_modulus_minus_one` (notice the change from lower equal to strictly lower).

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.11 The `field_t` multiplication by zero result not canonicalized to constant

Severity: Informational

Context: field.cpp#L214-L218

Description: Whenever you multiply a non-constant field by the constant zero, the code produces a `field_t` with:

```
result.additive_constant = additive_constant * other.additive_constant;
result.multiplicative_constant = multiplicative_constant * other.additive_constant;
result.witness_index = witness_index;
```

so `is_constant` remains `false` even though the represented value is exactly zero. Subsequent branches like in `assert_is_zero` or `operator+` will treat it as a variable, creating unnecessary gates.

Recommendation: We recommend that after any operation that yields `multiplicative_constant == 0`, it should turn into a `true` constant.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.12 Duplicate constraint in `invert`

Severity: Informational

Context: field.hpp#L269

Description: The `invert` method calls division with a dividend set to the constant 1. This creates 2 constraints: the first proves that the divisor is not zero, and the second one proves the result of the division. But when the dividend is a constant 1, those two constraints are the same. Consider:

- Input = `w_1*3 + 5`.

The `assert_is_not_zero` generates the following constraint:

- `(w_1*3 + 5)*w_2 = 1 <=> 3*w_1*w_2 + 5*w_2 - 1 = 0` where `w_2` is the inverse of `w_1`.

Then `divide_no_zero_check` generates the following constraint for the result witness `w_3`:

- `(w_1*3 + 5)*w_3 = 1 <=> 3*w_3*w_1 + 5*w_3 - 1 = 0`.

The solution to the division is the inverse of the input, which we already got via the constrain in `assert_is_not_zero`.

Recommendation: The `operator/` check for the divisor to not be `0` is only required if the dividend can be `0`. If we know the dividend is not `0` (the case where it is a non-zero constant) we can skip that constraint.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.13 Unexpected random value in `test_slice_random`

Severity: Informational

Context: field.test.cpp#L843

Description: The test `test_slice_random` is expected to test the slice method with a random input that as 252 bits. To get this number a random value is generated and then it's expected to be masked, but the mask operator is a logical or instead of a bitwise or, leading to a boolean result that is 1 with high provability.

Recommendation: Replace the logical or by a bitwise or:

```
fr a_ = fr(engine.get_random_uint256() & ((uint256_t(1) << 252) - 1));
```

Notice the `&` vs `&&`.

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.

3.2.14 `test_slice_random` not covering the entire input space

Severity: Informational

Context: field.test.cpp#L843-L849

Description: The `test_slice_random` prepares an input that is 252 bits. But the `slice` method works with an input that is `253` bits. The slice method doesn't work with values that need more than `253` to avoid aliasing (multiple valid solutions that are congruent modulo p).

The `slice` method constraints:

- `lo < 2^lsb`.
- `mid < 2^(msb+1 - lsb)`.
- `hi < 2^(252 - msb)`.

Then the linear combination leads to the following amount of bits:

- `lsb + msb + 1 - lsb + 252 - msb = 253`.

Perhaps the constraint on the number of bits for `hi` in `slice` would be more clear like this:

```
hi_wit.create_range_constraint(253 - msb_plus_one, "slice: hi value too large.");
```

Recommendation: Mask the input to be 253 bits instead of 252 bits:

```
fr a_ = fr(engine.get_random_uint256() & ((uint256_t(1) << 253) - 1));
```

Calculate the high part correctly:

```
const uint256_t expected2 = (uint256_t(a_) >> (msb + 1)) & ((uint256_t(1) << (253 - msb - 1)) - 1);
```

Aztec Labs: Fixed in commit 4433c06a.

Cantina Managed: Fix verified.