



---

## **ZKsync protocol Security Review**

---

### **Auditors**

Guido Vranken, Lead Security Researcher

Blockdev, Security Researcher

Kyle Charbonnet, Security Researcher

**Report prepared by:** Lucas Goiriz

April 14, 2025

# Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>About Spearbit</b>   | <b>2</b> |
| <b>2</b> | <b>Introduction</b>   | <b>2</b> |
| <b>3</b> | <b>Risk classification</b>  | <b>2</b> |
| 3.1      | Impact . . . . .  | 2        |
| 3.2      | Likelihood . . . . .  | 2        |
| 3.3      | Action required for severity levels . . . . .   | 2        |
| <b>4</b> | <b>Executive Summary</b>  | <b>3</b> |
| <b>5</b> | <b>Findings</b>   | <b>4</b> |
| 5.1      | Informational . . . . .   | 4        |
| 5.1.1    | Redundant code in scalar multiplication due to bounds on $k_1, k_2$ . . . . .                         | 4        |
| 5.1.2    | Analysis of G2 point manipulation functions and their unsupported cases . . . . .                     | 19       |
| 5.1.3    | Improving Docs . . . . .  | 23       |
| 5.1.4    | Negate-and-subtract operation can be replaced with addition . . . . .                                 | 23       |
| 5.1.5    | multipairing_naive(...) isn't implemented correctly for NUM_PAIRINGS_IN_MULTIPAIRING<br>> 1 . . . . . | 24       |

# 1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at [spearbit.com](https://spearbit.com)

## 2 Introduction

Matter Labs is an engineering team passionate about liberty, blockchain, and math; also known as the inventors of ZKsync: the leading cluster of ZK rollups on Ethereum.

*Disclaimer:* This security review does not guarantee against a hack. It is a snapshot in time of ZKsync protocol according to the specific commit. Any modifications to the code will require a new security review.

## 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         |

### 3.1 Impact

- 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.

### 3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

### 3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

## 4 Executive Summary

Over the course of 17 days in total, [Matter Labs](#) engaged with [Spearbit](#) to review the [zksync-protocol](#) protocol. In this period of time a total of **5** issues were found.

### Summary

|                        |                                 |
|------------------------|---------------------------------|
| <b>Project Name</b>    | Matter Labs                     |
| <b>Repository</b>      | <a href="#">zksync-protocol</a> |
| <b>Commit</b>          | <a href="#">4c44b9f8</a>        |
| <b>Type of Project</b> | ZK, Infrastructure              |
| <b>Audit Timeline</b>  | Mar 22nd to Apr 8th             |

### Issues Found

| <b>Severity</b>   | <b>Count</b> | <b>Fixed</b> | <b>Acknowledged</b> |
|-------------------|--------------|--------------|---------------------|
| Critical Risk     | 0            | 0            | 0                   |
| High Risk         | 0            | 0            | 0                   |
| Medium Risk       | 0            | 0            | 0                   |
| Low Risk          | 0            | 0            | 0                   |
| Gas Optimizations | 0            | 0            | 0                   |
| Informational     | 5            | 0            | 0                   |
| <b>Total</b>      | <b>5</b>     | <b>0</b>     | <b>0</b>            |

## 5 Findings

### 5.1 Informational

#### 5.1.1 Redundant code in scalar multiplication due to bounds on $k_1, k_2$

**Severity:** Informational

**Context:** (No context files were provided by the reviewer)

**Description:** With  $0 \leq k < n$ :

$$0 \leq k_1 \leq 166069116403752002167832803607207952478$$

$$-160042871798160843020181221280528662475 \leq k_2 \leq 4965661367192848883$$

```
from z3 import *

def find_solution(k1, expr):
    solver = Solver()

    lambd = Int('lambd')
    a1 = Int('a1')
    b1 = Int('b1')
    a2 = Int('a2')
    b2 = Int('b2')
    n = Int('n')
    g1 = Int('g1')
    g2 = Int('g2')
    c1 = Int('c1')
    c2 = Int('c2')
    q1 = Int('q1')
    q2 = Int('q2')
    k2 = Int('k2')
    k2_lambda = Int('k2_lambda')
    k = Int('k')

    solver.add(lambd == 4407920970296243842393367215006156084916469457145843978461)
    solver.add(a1 == 0x89d3256894d213e3)
    solver.add(b1 == -0x6f4d8248eeb859fc8211bbeb7d4f1128)
    solver.add(a2 == 0x6f4d8248eeb859fd0be4e1541221250b)
    solver.add(b2 == 0x89d3256894d213e3)
    solver.add(n == 21888242871839275222246405745257275088548364400416034343698204186575808495617)
    solver.add(g1 == 782660544089080853078787955015628534157)
    solver.add(g2 == 0x2d91d232ec7e0b3d7)
    solver.add(c1 == (g2 * k) / (2**256))
    solver.add(c2 == (g1 * k) / (2**256))
    solver.add(q1 == c1 * b1)
    solver.add(q2 == -(c2 * b2))
    solver.add(k2 == q2 - q1)
    solver.add(k2_lambda == (k2 * lambd) % n)
    solver.add(k1 == k - k2_lambda)
    solver.add(k >= 0)
    solver.add(k < n)
    solver.add(eval(expr))

    res = solver.check()
    if res == sat:
        m = solver.model()
        print("Solution found:")
        print(m)
```

```

elif res == unsat:
    print("Solution does not exist")
else:
    print("Unknown if solution exists")

exprs = [
    'k1 == 0',
    'k1 < 0',
    'k1 == 166069116403752002167832803607207952478',
    'k1 > 166069116403752002167832803607207952478',
    'k2 == -160042871798160843020181221280528662475',
    'k2 < -160042871798160843020181221280528662475',
    'k2 == 4965661367192848883',
    'k2 > 4965661367192848883',
]
for e in exprs:
    k1 = Int('k1')
    print('Solving for', e)
    find_solution(k1, e)
    print()

```

Output should be similar to:

```

Solving for k1 == 0
Solution found:
[q1 = 0,
 b1 = -147946756881789319000765030803803410728,
 c1 = 0,
 g1 = 782660544089080853078787955015628534157,
 a2 = 147946756881789319010696353538189108491,
 c2 = 0,
 k2_lambda = 0,
 lambda = 4407920970296243842393367215006156084916469457145843978461,
 g2 = 52538187511802934231,
 b2 = 9931322734385697763,
 k1 = 0,
 a1 = 9931322734385697763,
 k = 0,
 q2 = 0,
 k2 = 0,
 n = 21888242871839275222246405745257275088548364400416034343698204186575808495617]

Solving for k1 < 0
Solution does not exist

Solving for k1 == 166069116403752002167832803607207952478
Solution found:
[g1 = 782660544089080853078787955015628534157,
 q1 = -1469306990098747947415140152708122255298613926957756390736,
 c1 = 9931322734385697762,
 a2 = 147946756881789319010696353538189108491,
 g2 = 52538187511802934231,
 lambda = 4407920970296243842393367215006156084916469457145843978461,
 c2 = 147946756881789319000765030803803410726,
 b2 = 9931322734385697763,
 k2_lambda = 21888242871839275222246405745257275088252470886652455705666880156765044580872,
 k1 = 166069116403752002167832803607207952478,
 a1 = 9931322734385697763,
 n = 21888242871839275222246405745257275088548364400416034343698204186575808495617,
 k = 21888242871839275222246405745257275088418540003056207707834712960372252533350,
 q2 = -1469306990098747947563086909589911574279516312292788405938,
 b1 = -147946756881789319000765030803803410728,

```

```

k2 = -147946756881789318980902385335032015202]

Solving for k1 > 166069116403752002167832803607207952478
Solution does not exist

Solving for k2 == -160042871798160843020181221280528662475
Solution found:
[q1 = -1469306990098747947267193395826332936297848896153952980008,
k2_lambda = 21888242871839275220222640891709206871962320426167132852873346165044265606174,
a1 = 9931322734385697763,
c1 = 9931322734385697761,
n = 2188824287183927522246405745257275088548364400416034343698204186575808495617,
k1 = 18122359521962683156272859136894859594,
b1 = -147946756881789319000765030803803410728,
c2 = 147946756881789318987086022921494283441,
g1 = 782660544089080853078787955015628534157,
k2 = -160042871798160843020181221280528662475,
a2 = 147946756881789319010696353538189108491,
k = 21888242871839275220222640891709206871980442785689095536029619024181160465768,
g2 = 52538187511802934231,
lambda = 4407920970296243842393367215006156084916469457145843978461,
q2 = -1469306990098747947427236267624493779318030117434481642483,
b2 = 9931322734385697763]

Solving for k2 < -160042871798160843020181221280528662475
Solution does not exist

Solving for k2 == 4965661367192848883
Solution found:
[q1 = -443840270645367957002295092411410232184,
b2 = 9931322734385697763,
c1 = 3,
k2_lambda = 6611881455444365763516077444068339467869356008949671413446,
a1 = 9931322734385697763,
n = 2188824287183927522246405745257275088548364400416034343698204186575808495617,
c2 = 44690952304735639927,
k1 = 128405895564566517632095272875416729992,
b1 = -147946756881789319000765030803803410728,
g1 = 782660544089080853078787955015628534157,
k2 = 4965661367192848883,
a2 = 147946756881789319010696353538189108491,
k = 6611881455444365763644483339632905985501451281825088143438,
q2 = -443840270645367956997329431044217383301,
g2 = 52538187511802934231,
lambda = 4407920970296243842393367215006156084916469457145843978461]

Solving for k2 > 4965661367192848883
Solution does not exist

```

I am using Z3 version 4.14.1.0 on Linux x64. Older versions might be incapable of solving for these constraints. This implies that:

- $k_1$  is never negative.
- Neither  $\text{abs}(k_1)$  or  $\text{abs}(k_2)$  can be more than 127 bits.

A patch that removes redundant code and adds tests:

```

diff --git a/crates/zkevm_circuits/src/bn254/ec_mul/implementation.rs
↪ b/crates/zkevm_circuits/src/bn254/ec_mul/implementation.rs
index 91140c6..fbd0fb5 100644
--- a/crates/zkevm_circuits/src/bn254/ec_mul/implementation.rs
+++ b/crates/zkevm_circuits/src/bn254/ec_mul/implementation.rs

```

```

@@ -20,7 +20,7 @@ use super::*;

// Width 4 windowed multiplication parameters
const WINDOW_WIDTH: usize = 4;
-const NUM_MULTIPLICATION_STEPS_FOR_WIDTH_4: usize = 33;
+const NUM_MULTIPLICATION_STEPS_FOR_WIDTH_4: usize = 32;
const PRECOMPUTATION_TABLE_SIZE: usize = (1 << WINDOW_WIDTH) - 1;

/// BETA parameter such that  $\phi(x, y) = (\beta \cdot x, y)$ 
@@ -139,7 +139,6 @@ where
pub struct ScalarDecomposition<F: SmallField> {
    pub k1: BN256ScalarNNField<F>,
    pub k2: BN256ScalarNNField<F>,
-    pub k1_was_negated: Boolean<F>,
    pub k2_was_negated: Boolean<F>,
}

@@ -215,21 +214,10 @@ where
    let mut k1 = scalar.sub(cs, &mut k2_times_lambda);
    k1.normalize(cs);

-    let k1_u256 = convert_field_element_to_uint256(cs, k1.clone());
-    let k2_u256 = convert_field_element_to_uint256(cs, k2.clone());

    let low_pow_2_128 = pow_2_128.to_low();

-    // Selecting between k1 and -k1 in Fq
-    let (_, k1_out_of_range) = low_pow_2_128.overflowing_sub(cs, &k1_u256);
-    let k1_negated = k1.negated(cs);
-    let k1 = <BN256ScalarNNField<F> as NonNativeField<F, BN256Fr>>::conditionally_select(
-        cs,
-        k1_out_of_range,
-        &k1_negated,
-        &k1,
-    );

    // Selecting between k2 and -k2 in Fq
    let (_, k2_out_of_range) = low_pow_2_128.overflowing_sub(cs, &k2_u256);
    let k2_negated = k2.negated(cs);
@@ -243,7 +231,6 @@ where
    Self {
        k1,
        k2,
-        k1_was_negated: k1_out_of_range,
        k2_was_negated: k2_out_of_range,
    }
}

@@ -288,16 +275,6 @@ where

// we also know that we will multiply k1 by points, and k2 by their endomorphisms, and if they were
// negated above to fit into range, we negate bases here
- for (_, y) in table.iter_mut() {
-     let negated = y.negated(cs);
-     *y = Selectable::conditionally_select(
-         cs,
-         scalar_decomposition.k1_was_negated,
-         &negated,
-         &*y,
-     );
- }

for (_, y) in endomorphisms_table.iter_mut() {

```



```

        let negated = y.negated(cs);
        *y = Selectable::conditionally_select(
@@ -399,18 +376,7 @@ fn to_width_4_window_form<F: SmallField, CS: ConstraintSystem<F>>(
    let byte_split_id = cs
        .get_table_id_for_marker::<ByteSplitTable<4>>()
        .expect("table should exist");
-   let mut result = Vec::with_capacity(33);
-   // special case
-   {
-       let highest_word = limited_width_scalar.limbs[8];
-       let word = unsafe { UInt16::from_variable_unchecked(highest_word) };
-       let [high, low] = word.to_be_bytes(cs);
-       Num::enforce_equal(cs, &high.into_num(), &zero_num);
-       let [l, h] = cs.perform_lookup::<1, 2>(byte_split_id, &[low.get_variable()]);
-       Num::enforce_equal(cs, &Num::from_variable(h), &zero_num);
-       let l = Num::from_variable(l);
-       result.push(l);
-   }
+   let mut result = Vec::with_capacity(NUM_MULTIPLICATION_STEPS_FOR_WIDTH_4);

    for word in limited_width_scalar.limbs[..8].iter().rev() {
        let word = unsafe { UInt16::from_variable_unchecked(*word) };
diff --git a/crates/zkevm_circuits/src/bn254/tests/ec_mul.rs
↪ b/crates/zkevm_circuits/src/bn254/tests/ec_mul.rs
index 36c4616..5681891 100644
--- a/crates/zkevm_circuits/src/bn254/tests/ec_mul.rs
+++ b/crates/zkevm_circuits/src/bn254/tests/ec_mul.rs
@@ -184,13 +184,11 @@ pub mod test {
    // Actual:
    let decomposition = ScalarDecomposition::from(cs, &mut k, &scalar_params);
    let k1 = decomposition.k1.witness_hook(cs).unwrap().get();
-   let k1_was_negated = decomposition.k1_was_negated.witness_hook(cs).unwrap();
    let k2 = decomposition.k2.witness_hook(cs).unwrap().get();
    let k2_was_negated = decomposition.k2_was_negated.witness_hook(cs).unwrap();

    // Asserting:
    assert_eq!(k1, expected_k1);
-   assert_eq!(k1_was_negated, test.k1_negated);
    assert_eq!(k2, expected_k2);
    assert_eq!(k2_was_negated, test.k2_negated);

diff --git a/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/decomposition_tests.json
↪ b/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/decomposition_tests.json
index c84d74a..35b8b50 100644
--- a/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/decomposition_tests.json
+++ b/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/decomposition_tests.json
@@ -1,73 +1,129 @@
{
  "tests": [
    {
-     "k": "15310371241001622231792573671034611146751398180949444714681938262507259021376",
-     "k1": "31493387514781804819799587401213100432",
-     "k1_negated": false,
-     "k2": "151157157702727883088007207684649152593",
+     "k": "83968325905005542796731739941211061641431135680830130355353111482392216598981",
+     "k1": "87665704980795851588939397415006747113",
+     "k2": "52199859085699918696249724914764548231",
      "k2_negated": true
    },
    {
-     "k": "5968109632458944725585630137251814127900275934231084115272299218230903918742",
-     "k1": "148368310044984819298439648426363465247",

```

```

-     "k1_negated": false,
-     "k2": "70436765875676106276307438959739202343",
+     "k": "115792089237316195423570985008687907853269984665640564039457584007913129639935",
+     "k1": "95870001225982310100007781552424587595",
+     "k2": "63990262992893560344225959425260160623",
+     "k2_negated": true
  },
  {
-     "k": "1742498084139980620251305439875895103323009640850145693178755079904652572667",
-     "k1": "79390217055891588613575440186243325333",
-     "k1_negated": false,
-     "k2": "43691974796562340303443556971705007273",
+     "k": "21888242871839275222246405745257275088418540003056207707834712960372252533350",
+     "k1": "166069116403752002167832803607207952478",
+     "k2": "147946756881789318980902385335032015202",
+     "k2_negated": true
  },
  {
-     "k": "12435158121183200268140084484429025814293561617370699983291622298140646353039",
-     "k1": "52263466019978366513411579391808259690",
-     "k1_negated": false,
-     "k2": "147621535299651720294363510087868210346",
+     "k": "6611881455444365763644483339632905985501451281825088143438",
+     "k1": "128405895564566517632095272875416729992",
+     "k2": "4965661367192848883",
+     "k2_negated": false
  },
  {
+     "k": "2188824287183927522022640891709206871998586958063652822062594796206231189982",
+     "k1": "36266531896519969189248631161965583808",
+     "k2": "160042871798160843020181221280528662475",
+     "k2_negated": true
  },
  {
+     "k": "21888242871839275222246405745257275088548364400416034343668410218372651402328",
+     "k1": "147946756881789318990833708069417712965",
+     "k2": "147946756881789318990833708069417712965",
+     "k2_negated": true
  },
  {
+     "k": "147946756881789319020627676272574806254",
+     "k1": "9931322734385697763",
+     "k2": "9931322734385697763",
+     "k2_negated": true
  },
  {
-     "k": "17338514797382288172543346539328634339587720299195401471586177872534374281412",
-     "k1": "69637526295134927809587812840081959570",
-     "k1_negated": false,
-     "k2": "116242088082134145330112269101732161198",
+     "k": "115792089237316195423570985008687907853122037908758774720446887654374940531445",
+     "k1": "95870001225982310100007781552424587596",
+     "k2": "63990262992893560334294636690874462860",
+     "k2_negated": true
  },
  {
-     "k": "4424259225288216396589498511088720282107929032919356608555421459560833194920",
-     "k1": "91771513616118559935694594721410146722",
-     "k1_negated": false,
-     "k2": "57839291307811184798433275825492233393",
+     "k": "21888242871839275222246405745257275088418540003056207707834712960372252533351",
+     "k1": "18122359521962683157136450069018843988",

```

```

+         "k2": "147946756881789318990833708069417712965",
+         "k2_negated": true
+     },
+     {
-         "k": "12703330644849743193798861201554125235863198764416081532769166352996028110428",
-         "k1": "70304310444027999311628774683115887163",
-         "k1_negated": false,
-         "k2": "18942669441150157207684220257508261454",
+         "k": "21888242871839275220222640891709206872012645081782417699208530339517524196747",
+         "k1": "50324655615284846325252851738872892810",
+         "k2": "12096114916371524019416190476725251747",
+         "k2_negated": true
+     },
+     {
-         "k": "8779674889540996071735414813021738730093065101253818851641949545992578872205",
-         "k1": "49277213207367892196602854559228360108",
-         "k1_negated": false,
-         "k2": "146797052628359847097274906309483172946",
+         "k": "115792089237316195421367024523539785932055156885762928710957100247304766925457",
+         "k1": "151699207292319683563106926286078416593",
+         "k2": "63990262992893560339260298058067311742",
+         "k2_negated": true
+     },
+     {
-         "k": "10908826393692157466283479854939993281909445730438484613349035400366196560258",
-         "k1": "144199595830261842889193902907642090635",
-         "k1_negated": false,
-         "k2": "152612964910770730162553831008213365471",
+         "k": "115792089237316195416959103553243542089605960464690435252572057341546526934865",
+         "k1": "95870001225982310094533477673682404462",
+         "k2": "63990262992893560339260298058067311743",
+         "k2_negated": true
+     },
+     {
+         "k": "21888242871839275220222640891709206871980442785689095536029619024181160465768",
+         "k1": "18122359521962683156272859136894859594",
+         "k2": "160042871798160843020181221280528662475",
+         "k2_negated": true
+     },
+     {
+         "k": "2188824287183927522246405745257275088418540003056207707834712960372252533351",
+         "k1": "18122359521962683157136450069018843988",
+         "k2": "147946756881789318990833708069417712965",
+         "k2_negated": true
+     },
+     {
+         "k": "158411906564648057531936279836761707509396003214563100670",
+         "k1": "10633823966279326983230456482242756609",
+         "k2": "10633823966279326994231158980716092773",
+         "k2_negated": true
+     },
+     {
+         "k": "316823813129296114915925802791734096008095652890937092848",
+         "k1": "21267647932558653966460912964485513217",
+         "k2": "21267647932558653978530995227046487783",
+         "k2_negated": true
+     },
+     {
+         "k": "5041568596554836071929325306826045638911268055851339947173",
+         "k1": "42535295865117307932921825928971026432",
+         "k2": "42535295865117307937199344985321580039",
+         "k2_negated": true
+     }

```

```

+     },
+     {
+         "k": "7879176707961550222587993627708118575858953206360663350870",
+         "k1": "85070591730234615865843651857942052864",
+         "k2": "85070591730234615869433028603450311197",
+         "k2_negated": true
+     },
+     {
+         "k": "85070591730234615865843651857942052863",
+         "k1": "85070591730234615865843651857942052863",
+         "k2": "0",
+         "k2_negated": false
+     },
+     {
+         "k": "15821689026819535985381806513542673298904692023783692940182535323294479792772",
+         "k1": "13099559360135235052400423830238750792",
+         "k2": "85070591730234615865843651857942052863",
+         "k2_negated": true
+     },
+     {
+         "k": "18719334488860627500369957823428539212555536533361315655716333592432086088405",
+         "k1": "78693297012568138413531716443444140114",
+         "k1_negated": false,
+         "k2": "84127714618519115178173319957441990163",
+         "k": "15821689026819535985381806513542673298976663056153792320995978551322183094843",
+         "k1": "85070591730234615865843651857942052863",
+         "k2": "85070591730234615865843651857942052863",
+         "k2_negated": true
+     }
+ ]
diff --git a/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/ecmul_tests.json
↪ b/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/ecmul_tests.json
index 208b37a..548b5b2 100644
--- a/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/ecmul_tests.json
+++ b/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/ecmul_tests.json
@@ -2,13 +2,233 @@
  "tests": [
    {
      "point": {
-       "x": "14097009101881959050629049093828651584107527035947797050538346806411625303116",
-       "y": "6928765890834363798765710535428389975333897900712784906653282110125786142062"
+       "x": "1",
+       "y": "2"
      },
-     "scalar": "13650076562025738285589406854928154499107354233219361696036823113035450875054",
+     "scalar": "83968325905005542796731739941211061641431135680830130355353111482392216598981",
      "expected": {
-       "x": "11299373567935086735078551232826217925502745784801646334636571278818215743539",
-       "y": "18836805603793619172102959251973968032345476967462197112983508530862248992053"
+       "x": "139251042355124443337096206823249605394137811693364370970845201517602958265",
+       "y": "4320624229966780452519355006993679778917586788720681614271924571570292308216"
      }
    },
    {
      "point": {
+       "x": "1",
+       "y": "2"
      },
      "scalar": "115792089237316195423570985008687907853269984665640564039457584007913129639935",
      "expected": {
+       "x": "21415159568991615317144600033915305503576371596506956373206836402282692989778",
+       "y": "8573070896319864868535933562264623076420652926303237982078693068147657243287"

```

```

+     }
+ },
+ {
+     "point": {
+         "x": "1",
+         "y": "2"
+     },
+     "scalar": "21888242871839275222246405745257275088418540003056207707834712960372252533350",
+     "expected": {
+         "x": "5793346190569007546688430134772645231473610751713945182487469508120330077373",
+         "y": "13543497425240060327382100399841884843806450700078130961320280218982647557466"
+     }
+ },
+ {
+     "point": {
+         "x": "1",
+         "y": "2"
+     },
+     "scalar": "6611881455444365763644483339632905985501451281825088143438",
+     "expected": {
+         "x": "17843796094153804475475462008069489967746435921297757021583166015102577531041",
+         "y": "140878724482851916862756513603124964463955335096561573142385408205751767103"
+     }
+ },
+ {
+     "point": {
+         "x": "1",
+         "y": "2"
+     },
+     "scalar": "21888242871839275220222640891709206871998586958063652822062594796206231189982",
+     "expected": {
+         "x": "12787867861020980678117956101723515349350391304192238540091144947347208088030",
+         "y": "13240302848311906527618553598894808278134638473061458549401574734857896047383"
+     }
+ },
+ {
+     "point": {
+         "x": "1",
+         "y": "2"
+     },
+     "scalar": "21888242871839275222246405745257275088548364400416034343668410218372651402328",
+     "expected": {
+         "x": "13194957371956931795709136998963250104760578165429223396476574392416761212575",
+         "y": "10065223513748416994106058133200810928700988293106150653510602746184807367191"
+     }
+ },
+ {
+     "point": {
+         "x": "1",
+         "y": "2"
+     },
+     "scalar": "147946756881789319020627676272574806254",
+     "expected": {
+         "x": "15361444639272000542079751084167515413400830796534362690103245709153005741604",
+         "y": "18893506883835740211889523270233281666885803481890970229437899355752901794202"
+     }
+ },
+ {
+     "point": {
+         "x": "1",
+         "y": "2"
+     },

```

```

+         "scalar": "115792089237316195423570985008687907853122037908758774720446887654374940531445",
+         "expected": {
+             "x": "7038363397236725556503034901916614878198259273625847771245278376723832468004",
+             "y": "11204263953223425886566292744655262591617732423796185120873217439982788330490"
+         },
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "21888242871839275222246405745257275088418540003056207707834712960372252533351",
+         "expected": {
+             "x": "10638222009551650162368399862989083299801515940505608772354555385850286439155",
+             "y": "20264529763687316034550287822991009082897970805969510880974535001006101257711"
+         },
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "21888242871839275220222640891709206872012645081782417699208530339517524196747",
+         "expected": {
+             "x": "9146422971183786756756061111970316211503611979370881413983665809450063177295",
+             "y": "21208322345121820858211404434166058876784291431720158714023298330671280668909"
+         },
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "115792089237316195421367024523539785932055156885762928710957100247304766925457",
+         "expected": {
+             "x": "13618646154367749796682906406458688511315473803005799913151151192028419080222",
+             "y": "11347701592262166697632660044916842096697633971385628841428848694897380621766"
+         },
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "115792089237316195416959103553243542089605960464690435252572057341546526934865",
+         "expected": {
+             "x": "10937923587686763230563875383359688295454426111030133206869199507050204106336",
+             "y": "2137810684820734425049242595126671500809555185843198473106273074850790521224"
+         },
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "21888242871839275220222640891709206871980442785689095536029619024181160465768",
+         "expected": {
+             "x": "18542674179642048922513611322467513905368281385695829134081762017312758261785",
+             "y": "15471240729095354197326663506201396891828764720282822193502076500997755701145"
+         },
+     },
+     {

```

```

+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "2188824287183927522246405745257275088418540003056207707834712960372252533351",
+         "expected": {
+             "x": "1063822200955165016236839986298908329980151594050560877235455385850286439155",
+             "y": "20264529763687316034550287822991009082897970805969510880974535001006101257711"
+         }
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "158411906564648057531936279836761707509396003214563100670",
+         "expected": {
+             "x": "19471846276042019603680307584544274325911846066163010374665343304981835897092",
+             "y": "18277112207102707428949791540633704169166384232791943847157801788808409924510"
+         }
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "316823813129296114915925802791734096008095652890937092848",
+         "expected": {
+             "x": "861560855474980543379328681622940873093930894954072904007939373277023003172",
+             "y": "13131411589591604447560458598903522319996873699357120759858308275580014461636"
+         }
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "5041568596554836071929325306826045638911268055851339947173",
+         "expected": {
+             "x": "11555598236105980090162589741189089757835849069636286665040692481344335397266",
+             "y": "16640942933358602787022104241447655190220191833032909580740271414977865990616"
+         }
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "7879176707961550222587993627708118575858953206360663350870",
+         "expected": {
+             "x": "14440313714323323140880735412599476459069451606950106107834541783649243212096",
+             "y": "17130312510137748447107739009726880103141899543936393503461192541031503631347"
+         }
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "85070591730234615865843651857942052863",
+         "expected": {
+             "x": "11077178434411445329823712134344807181518152298871108954538612262175904114084",

```

```

+         "y": "135858357939193311837670889571226333613812204691702859091544528896092599086"
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "15821689026819535985381806513542673298904692023783692940182535323294479792772",
+         "expected": {
+             "x": "4841644965230227573421955367229416044157638307018458983642363070152904375193",
+             "y": "16261352950699406497876161800396881920746243631902584350908851266673499507745"
+         }
+     },
+     {
+         "point": {
+             "x": "1",
+             "y": "2"
+         },
+         "scalar": "15821689026819535985381806513542673298976663056153792320995978551322183094843",
+         "expected": {
+             "x": "9000403088489384882008020958707497528698045496224659779011370565349001666928",
+             "y": "232315067373539605880195713829063964658952398843235784555051705053165290120"
+         }
+     }
+ ]
diff --git a/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/mod.rs
↪ b/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/mod.rs
index 6a77604..4dca3cf 100644
--- a/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/mod.rs
+++ b/crates/zkevm_circuits/src/bn254/tests/json/ec_mul/mod.rs
@@ -14,7 +14,6 @@ pub struct DecompositionTestCase {
     pub k: String,
     pub k1: String,
     pub k2: String,
-    pub k1_negated: bool,
-    pub k2_negated: bool,
 }

```

A script to fill in tests (both `test_scalar_decomposition` and `test_width_4_multiplication`) for some corner cases:

```

# **/usr/bin/env python3:**

# Based on src/bn254/sage/scalar_decomposition.sage.py

# **pip install py-ecc:** from py_ecc.bn128 import FQ, multiply

# **Defining vectors (a1,b1) and (a2,b2):** a1 = 0x89d3256894d213e3
b1 = -0x6f4d8248eeb859fc8211bbeb7d4f1128
a2 = 0x6f4d8248eeb859fd0be4e1541221250b
b2 = 0x89d3256894d213e3

# **Defining some curve parameters:** n =
↪ 21888242871839275222246405745257275088548364400416034343698204186575808495617

g1 = 0x24ccf014a773d2cf7a7bd9d4391eb18d
g2 = 0x2d91d232ec7e0b3d7
lambda = 4407920970296243842393367215006156084916469457145843978461

def decompose_aztec(k):
    c1 = (g2 * k) >> 256

```



```

c2 = (g1 * k) >> 256

q1 = c1 * b1
q2 = -c2 * b2

k2 = q2 - q1

k2_lambda = k2 * lambda % n
k1 = k - k2_lambda

return k1, k2

tests = [
    # k1 larger than 2**127 (note: k is larger than n)
    {'k' : 83968325905005542796731739941211061641431135680830130355353111482392216598981,
     'k1' : 65664728615517825666739217235771825265732758906228898882683551957142432233964,
     'k2' : -52199859085699918696249724914764548231,
    },
    # Largest (?) possible value for k1 (with 0 < k < 2**256)
    {
        "k": 115792089237316195423570985008687907853269984665640564039457584007913129639935,
        "k1": 109441214359196376111232028726286375442985638760187943347611656390704041871934,
        "k2": -211937019874682879335059667494677873588,
    },
    # Largest possible value for k1 (with 0 < k < n)
    {
        "k": 2188824287183927522246405745257275088418540003056207707834712960372252533350,
        "k1": 166069116403752002167832803607207952478,
        "k2": -147946756881789318980902385335032015202,
    },
    # Largest possible value for k2 (with 0 < k < n)
    {
        'k' : 6611881455444365763644483339632905985501451281825088143438,
        'k1' : 128405895564566517632095272875416729992,
        'k2' : 4965661367192848883,
    },
    # Smallest possible value for k2 (with 0 < k < n)
    {
        'k' : 21888242871839275220222640891709206871998586958063652822062594796206231189982,
        'k1' : 36266531896519969189248631161965583808,
        'k2' : -160042871798160843020181221280528662475,
    },
    # abs(k1) == abs(k2) (with 0 < k < n)
    {
        'k' : 2188824287183927522246405745257275088548364400416034343668410218372651402328,
        'k1' : 147946756881789318990833708069417712965,
        'k2' : -147946756881789318990833708069417712965,
    },
    # Smallest k with abs(k1) == abs(k2) (with 0 < k < n)
    {
        "k": 147946756881789319020627676272574806254,
        "k1": 9931322734385697763,
        "k2": -9931322734385697763,
    },
    # Smallest possible q2 (-7772854454818568418389002068570701163563450858166938293028) with 0 < k
    ↪ < 2**256
    {
        "k": 115792089237316195423570985008687907853122037908758774720446887654374940531445,
        "k1": 109441214359196376111232028726286375442837692003306154028600960037165852763444,
        "k2": -211937019874682879335059667494677873588,
    },
]

```

```

# Smallest possible q2 (-1469306990098747947563086909589911574289447635027174103701) with 0 < k
↳ < n
{
  "k": 21888242871839275222246405745257275088418540003056207707834712960372252533351,
  "k1": 18122359521962683157136450069018843988,
  "k2": -147946756881789318990833708069417712965,
},
# Smallest possible q1 (-1469306990098747947415140152708122255298613926957756390736) with 0 < k
↳ < n
{
  "k": 21888242871839275220222640891709206872012645081782417699208530339517524196747,
  "k1": 50324655615284846325252851738872892810,
  "k2": -12096114916371524019416190476725251747,
},
# Smallest possible q1 (-7772854454818568418177065048696018284228391190672260419440) with 0 < k
↳ < 2**256
{
  "k": 115792089237316195421367024523539785932055156885762928710957100247304766925457,
  "k1": 109441214359196376111232028726286375442893521209372491402054127859165120894678,
  "k2": -63990262992893560339260298058067311742,
},
# Largest difference between q1 and q2 (211937019874682879340025328861870722471) with 0 < k <
↳ 2**256
{
  "k": 115792089237316195416959103553243542089605960464690435252572057341546526934865,
  "k1": 109441214359196376111232028726286375442837692003306154028595485733287110580310,
  "k2": -211937019874682879340025328861870722471,
},
# Largest difference between q1 and q2 (160042871798160843020181221280528662475) with 0 < k < n
{
  "k": 21888242871839275220222640891709206871980442785689095536029619024181160465768,
  "k1": 18122359521962683156272859136894859594,
  "k2": -160042871798160843020181221280528662475,
},
# Largest k2_lambda
↳ (21888242871839275222246405745257275088400417643534245024677576510303233689363) with 0 < k <
↳ n
{
  "k": 21888242871839275222246405745257275088418540003056207707834712960372252533351,
  "k1": 18122359521962683157136450069018843988,
  "k2": -147946756881789318990833708069417712965,
},
# Large k1, k2 (>= 2**123)
{
  "k": 158411906564648057531936279836761707509396003214563100670,
  "k1": 10633823966279326983230456482242756609,
  "k2": -10633823966279326994231158980716092773,
},
# Large k1, k2 (>= 2**124)
{
  "k": 316823813129296114915925802791734096008095652890937092848,
  "k1": 21267647932558653966460912964485513217,
  "k2": -21267647932558653978530995227046487783,
},
# Large k1, k2 (>= 2**125)
{
  "k": 5041568596554836071929325306826045638911268055851339947173,
  "k1": 42535295865117307932921825928971026432,
  "k2": -42535295865117307937199344985321580039,
},
# Large k1, k2 (>= 2**126)
{

```

```

        "k": 7879176707961550222587993627708118575858953206360663350870,
        "k1": 85070591730234615865843651857942052864,
        "k2": -85070591730234615869433028603450311197,
    },
    # k1 with largest popcount (126)
    {
        "k": 85070591730234615865843651857942052863,
        "k1": 85070591730234615865843651857942052863,
        "k2": 0,
    },
    # k2 with largest popcount (126)
    {
        "k": 15821689026819535985381806513542673298904692023783692940182535323294479792772,
        "k1": 13099559360135235052400423830238750792,
        "k2": -85070591730234615865843651857942052863,
    },
    # k1, k2 with largest popcount (126+126)
    {
        "k": 15821689026819535985381806513542673298976663056153792320995978551322183094843,
        "k1": 85070591730234615865843651857942052863,
        "k2": -85070591730234615865843651857942052863,
    },
]

def write_decomposition_tests():
    j = {}
    j['tests'] = []
    for t in tests:
        tj = {}
        k = t['k']
        k1, k2 = decompose_aztec(k % n)
        tj['k'] = str(k)
        tj['k1'] = str(k1)
        tj['k2'] = str(abs(k2))
        tj['k2_negated'] = k2 < 0
        j['tests'] += [ tj ]

    import json
    with open('src/bn254/tests/json/ec_mul/decomposition_tests.json', 'wb') as fp:
        fp.write(json.dumps(j, indent=4).encode('utf-8'))

def write_ecmul_tests():
    j = {}
    j['tests'] = []
    for t in tests:
        in_x, in_y = 1, 2
        A = [FQ(in_x), FQ(in_y)]
        out_x, out_y = multiply(A, t['k'])
        tj = {}
        k = t['k']
        tj['point'] = {}
        tj['point']['x'] = str(in_x)
        tj['point']['y'] = str(in_y)
        tj['scalar'] = str(k)
        tj['expected'] = {}
        tj['expected']['x'] = str(out_x)
        tj['expected']['y'] = str(out_y)
        j['tests'] += [ tj ]

    import json
    with open('src/bn254/tests/json/ec_mul/ecmul_tests.json', 'wb') as fp:
        fp.write(json.dumps(j, indent=4).encode('utf-8'))

for t in tests:
    k = t['k']
    k1, k2 = decompose_aztec(k)

```

```

assert(k1 == t['k1'])
assert(k2 == t['k2'])
assert((k1 + k2 * lambda) % n == k % n)

write_decomposition_tests()
write_ecmul_tests()

```

This is an optimization finding incidental to the security analysis. The validity of this finding depends on Z3 being correct. Resort to deeper analysis for stronger guarantees.

**Matter Labs:** Fixed in [PR 140](#).

### 5.1.2 Analysis of G2 point manipulation functions and their unsupported cases

**Severity:** Informational

**Context:** (No context files were provided by the reviewer)

**Description:** TwistedCurvePoint in alternative\_pairing.rs implements several G2 point manipulation functions. Analysis shows these have several notable limitations with regards to the input points they support.

| Function       | Operation | Used by                                     | Unsupported cases           |
|----------------|-----------|---|-----------------------------|
| negate         | -P        | Creation of q_negated_array, subgroup check | None                        |
| sub            | P-Q       | subgroup check                              | P=Q, P=-Q, P=O, Q=O         |
| add            | P+Q       | none  | P=Q, P=-Q, P=O, Q=O         |
| double         | P+P       | subgroup check                              | P=O, P.Y*2=0                |
| double_and_add | P+P+Q     | subgroup check                              | P=O, Q=O, P=Q, P=-Q, P+Q=-P |

- sub: Inputs whose affine X coordinate is the same (e.g. P=Q and P=-Q) cause division by zero:

```

let mut other_x_minus_this_x = other.x.sub(cs, &mut self.x);
let mut other_y_plus_this_y = other.y.add(cs, &mut self.y);
let mut lambda = other_y_plus_this_y.div(cs, &mut other_x_minus_this_x);

```

- add: Inputs whose affine X coordinate is the same (e.g. P=Q and P=-Q) cause division by zero:

```

let mut other_x_minus_this_x = other.x.sub(cs, &mut self.x);
let mut other_y_minus_this_y = other.y.sub(cs, &mut self.y);
let mut lambda = other_y_minus_this_y.div(cs, &mut other_x_minus_this_x);

```

Additionally, any infinity input may simply produce an incorrect result (I determined this can happen during randomized differential testing). This method is currently not used.

To assert algebraic equivalence between the implementation and the affine addition formula at <https://www.hyperelliptic.org/EFD/g1p/data/shortw/coordinates>:

```

q = 21888242871839275222246405745257275088696311157297823662689037894645226208583
Fq = GF(q)

K2.<x> = PolynomialRing(Fq)
Fq2.<u> = Fq.extension(x^2+1)

b = 3 / (u + 9)
E = EllipticCurve(Fq2, [0, b])

R.<x1,y1,x2,y2> = PolynomialRing(Fq2, 4)

```

```

def validate_add():
    if True:
        # https://www.hyperelliptic.org/EFD/g1p/data/shortw/coordinates
        # addition x = (y2-y1)^2/(x2-x1)^2-x1-x2
        # addition y = (2 x1+x2) (y2-y1)/(x2-x1)-(y2-y1)^3/(x2-x1)^3-y1
        new_x_canonical = (y2-y1)^2/(x2-x1)^2-x1-x2
        new_y_canonical = (2*x1+x2)*(y2-y1)/(x2-x1)-(y2-y1)^3/(x2-x1)^3-y1

    if True:
        # let mut other_x_minus_this_x = other.x.sub(cs, &mut self.x);
        other_x_minus_this_x = x2 - x1

        # let mut other_y_minus_this_y = other.y.sub(cs, &mut self.y);
        other_y_minus_this_y = y2 - y1

        # let mut lambda = other_y_minus_this_y.div(cs, &mut other_x_minus_this_x);
        lambda_impl = other_y_minus_this_y / other_x_minus_this_x

        # let mut lambda_squared = lambda.square(cs);
        lambda_squared = lambda_impl^2

        # let mut other_x_plus_this_x = other.x.add(cs, &mut self.x);
        other_x_plus_this_x = x2 + x1

        # let mut new_x = lambda_squared.sub(cs, &mut other_x_plus_this_x);
        new_x = lambda_squared - other_x_plus_this_x

        # let mut this_x_minus_new_x = self.x.sub(cs, &mut new_x);
        this_x_minus_new_x = x1 - new_x

        # let mut new_y = lambda.mul(cs, &mut this_x_minus_new_x);
        new_y = lambda_impl * this_x_minus_new_x

        # new_y = new_y.sub(cs, &mut self.y);
        new_y = new_y - y1

        # Verify the implementations match
        assert(bool(new_x_canonical == new_x))
        assert(bool(new_y_canonical == new_y))
        print('add ok')
    validate_add()

```

- double\_and\_add: This implementation is based on <https://arxiv.org/pdf/math/0208038> which explicitly lists (in Appendix A) separate handling of infinity inputs ( $P=O$ ,  $Q=O$ ), inputs with the same X coordinate ( $P=Q$ ,  $P=-Q$ ) and the point tripling case ( $P+Q=-P$ ). These cases are unsupported by the current implementation.

To assert algebraic equivalence between the implementation and the algorithm in the paper (minus the special case handling):

```

q = 21888242871839275222246405745257275088696311157297823662689037894645226208583 # BN curve
↳ group order
Fq = GF(q)

# **Define the quadratic extension field:** K2.<x> = PolynomialRing(Fq)
Fq2.<u> = Fq.extension(x^2+1) # Quadratic extension with i^2 = -1

# **Define the G2 curve parameters:** a1 = 0
a2 = 0
a3 = 0
b2 = 3 / (u + 9) # Twisted curve parameter
E2 = EllipticCurve(Fq2, [0, b2]) # y^2 = x^3 + b2

```

```

# **Create symbolic variables for a generic point on G2:** R.<x1,y1,x2,y2> = PolynomialRing(Fq2,
↳ 4)

def validate_double_and_add():
    if True:
        #  $\lambda = (y - y)/(x - x)$ 
        lambda_1 = (y2-y1)/(x2-x1)

        #  $x = \lambda(\lambda + a) - a - x - x$ 
        x3 = lambda_1*(lambda_1+a1) - a2 - x1 - x2

        #  $\lambda = (ax + a + 2y)/(x - x) - \lambda$ 
        lambda_2 = (a1*x3+a3+2*y1)/(x1 - x3) - lambda_1

        #  $x = \lambda(\lambda + a) - a - x - x$ 
        new_x_canonical = lambda_2*(lambda_2+a1) - a2 - x1 - x3
        #  $y = \lambda(x - x) - ax - a - y$ 
        new_y_canonical = lambda_2*(x1 - new_x_canonical) - a1*new_x_canonical - a3 - y1
    if True:
        # let mut other_x_minus_this_x = other.x.sub(cs, &mut self.x);
        other_x_minus_this_x = x2 - x1

        # let mut other_y_minus_this_y = other.y.sub(cs, &mut self.y);
        other_y_minus_this_y = y2 - y1

        # let mut lambda = other.y_minus_this_y.div(cs, &mut other_x_minus_this_x);
        lambda_impl = other_y_minus_this_y / other_x_minus_this_x

        # let mut lambda_squared = lambda.square(cs);
        lambda_squared = lambda_impl^2

        # let mut other_x_plus_this_x = other.x.add(cs, &mut self.x);
        other_x_plus_this_x = x2 + x1

        # let mut new_x = lambda_squared.sub(cs, &mut other_x_plus_this_x);
        new_x = lambda_squared - other_x_plus_this_x

        # let mut new_x_minus_this_x = new_x.sub(cs, &mut self.x);
        new_x_minus_this_x = new_x - x1

        # let mut two_y = self.y.double(cs);
        two_y = y1 * 2

        # let mut t0 = two_y.div(cs, &mut new_x_minus_this_x);
        t0 = two_y / new_x_minus_this_x

        # let mut t1 = lambda.add(cs, &mut t0);
        t1 = lambda_impl + t0

        # let mut new_x_plus_this_x = new_x.add(cs, &mut self.x);
        new_x_plus_this_x = new_x + x1

        # let mut new_x = t1.square(cs);
        new_x = t1^2

        # new_x = new_x.sub(cs, &mut new_x_plus_this_x);
        new_x = new_x - new_x_plus_this_x

        # let mut new_x_minus_x = new_x.sub(cs, &mut self.x);
        new_x_minus_x = new_x - x1

        # let mut new_y = t1.mul(cs, &mut new_x_minus_x);

```

```

    new_y = t1 * new_x_minus_x

    # new_y = new_y.sub(cs, &mut self.y);
    new_y = new_y - y1

    # Verify the implementations match
    assert(bool(new_x_canonical == new_x))
    assert(bool(new_y_canonical == new_y))
    print('double-and-add ok')

validate_double_and_add()

```

- double: Point at infinity ( $Y=0$ ) and any other (potential) case where  $Y^2=0$  causes division by zero:

```

let mut two_y = self.y.double(cs);
let mut lambda = x_squared_3.div(cs, &mut two_y);

```

To assert algebraic equivalence between the implementation and the affine doubling formula at <https://www.hyperelliptic.org/EFD/g1p/data/shortw/coordinates>:

```

q = 21888242871839275222246405745257275088696311157297823662689037894645226208583
Fq = GF(q)

K2.<x> = PolynomialRing(Fq)
Fq2.<u> = Fq.extension(x^2+1)

a = 0
b = 3 / (u + 9)
E = EllipticCurve(Fq2, [0, b])

R.<x1,y1> = PolynomialRing(Fq2, 2)

def validate_double():
    # https://www.hyperelliptic.org/EFD/g1p/data/shortw/coordinates
    # doubling x = (3 x1^2+a)^2/(2 y1)^2-x1-x1
    # doubling y = (2 x1+x1) (3 x1^2+a)/(2 y1)-(3 x1^2+a)^3/(2 y1)^3-y1
    if True:
        new_x_canonical = (3*x1^2+a)^2/(2*y1)^2-x1-x1
        new_y_canonical = (2*x1+x1)*(3*x1^2+a)/(2*y1)-(3*x1^2+a)^3/(2*y1)^3-y1

    # alternative_pairing.rs TwistedCurvePoint<F>::double<CS: ConstraintSystem<F>>(&mut self, cs:
    ↪ &mut CS)
    if True:
        # let mut x_squared = self.x.square(cs);
        x_squared = x1^2

        # let mut x_squared_3 = x_squared.double(cs);
        x_squared_3 = x_squared * 2

        # x_squared_3 = x_squared_3.add(cs, &mut x_squared);
        x_squared_3 = x_squared_3 + x_squared

        # let mut two_y = self.y.double(cs);
        two_y = 2 * y1

        # let mut lambda = x_squared_3.div(cs, &mut two_y);
        lambda_impl = x_squared_3 / two_y

        # let mut lambda_squared = lambda.square(cs);
        lambda_squared = lambda_impl^2

```

```

    # let mut two_x = self.x.double(cs);
    two_x = 2 * x1

    # let mut new_x = lambda_squared.sub(cs, &mut two_x);
    new_x = lambda_squared - two_x

    # let mut x_minus_new_x = self.x.sub(cs, &mut new_x);
    x_minus_new_x = x1 - new_x

    # let mut new_y = x_minus_new_x.mul(cs, &mut lambda);
    new_y = x_minus_new_x * lambda_impl

    # new_y = new_y.sub(cs, &mut self.y);
    new_y = new_y - y1

    assert(bool(new_x_canonical == new_x))
    assert(bool(new_y_canonical == new_y))
    print('double ok')
    validate_double()

```

The implementation requires changes if used with a curve whose a parameter is nonzero.

**Recommendation:** My analysis shows there is currently probably no scenario where a point function will be invoked with an unsupported input point. Nonetheless, my suggestion would be to either implement complete point transformation formula's (which support every conceivable input point, at a higher computational cost) or to clearly annotate existing functions with their limitations, add debug asserts for unsupported cases and rigorously demonstrate (preferably prove) the range of possible inputs to these functions.

### 5.1.3 Improving Docs

**Severity:** Informational

**Context:** (No context files were provided by the reviewer)

**Description:** The current code has great inline comments, but overall I feel that including separate docs files would vastly improve the security of the code as it:

1. Helps with future audits for auditors to quickly ramp up and compare the code to the docs.
2. Helps with changes to the code as devs can refer to the docs for parts that may have been untouched for a while, and out of mind.
3. Helps with new devs to ramp up to the codebase in a secure way.

**Recommendation:** Including a doc that goes over all algorithms involved and includes references that point to the original algorithm would be very helpful. Perhaps clarifying in the doc what features are currently enabled would be great as well (for example, explaining that the code only supports a single pairing at the moment, but has code built-in to help enable multi-pairings in the future).

**Matter Labs:** Acknowledged. We will be postponing the documentation improvements for now.

### 5.1.4 Negate-and-subtract operation can be replaced with addition

**Severity:** Informational

**Context:** [implementation.rs#L205-L210](#)

**Description:**  $k_2$  here is calculated as  $q_2 - q_1$  after negating  $q_1$ .  $k_2$  can be calculated as  $q_2 + q_1$  if  $q_1$  isn't negating, optimizing.

**Recommendation:**



```

- let mut q1 = q1.negated(cs);
  let mut q2 = c2.mul(cs, &mut b2);
  let mut q2 = q2.negated(cs);

  // k2 <- q2 - q1
- let mut k2 = q2.sub(cs, &mut q1);
+ let mut k2 = q2.add(cs, &mut q1);

```

#### 5.1.5 multipairing\_naive(...) isn't implemented correctly for NUM\_PAIRINGS\_IN\_MULTIPAIRING > 1

**Severity:** Informational

**Context:** [alternative\\_pairing.rs#L1589-L1596](#)

**Description:** If NUM\_PAIRINGS\_IN\_MULTIPAIRING > 1, multipairing\_naive(...) is supposed to support multipairing. However, the function panics for this case. Additionally, if any point is infinity, the function returns result as 1. If there are more than one pairing operations, the output should depend on the remaining pairing operations.

**Proof of Concept:** This test panics in enforce\_pass\_through\_point:

```

**[test]** fn test_multipairing_2_elements() {
  /* Be sure to define NUM_PAIRINGS_IN_MULTIPAIRING to 2 */
  assert_eq!(NUM_PAIRINGS_IN_MULTIPAIRING, 2);

  let mut owned_cs = cs_geometry();
  let cs = &mut owned_cs;

  let params = RnsParams::create();
  let params = std::sync::Arc::new(params);

  let p1 = G1::one();
  let g1_1 = AffinePoint::allocate(cs, p1.into_affine(), &params);

  let p2 = G1::one();
  let g1_2 = AffinePoint::allocate(cs, p2.into_affine(), &params);

  let q1 = G2::one();
  let g2_1 = TwistedCurvePoint::allocate(cs, q1.into_affine(), &params);

  let q2 = G2::one();
  let g2_2 = TwistedCurvePoint::allocate(cs, q2.into_affine(), &params);

  let mut pairing_inputs = vec![(g1_1, g2_1), (g1_2, g2_2)];
  multipairing_naive(cs, &mut pairing_inputs);
}

```

**Recommendation:** Assert NUM\_PAIRINGS\_IN\_MULTIPAIRING is 1 in multipairing\_naive(...).