ZKsync protocol Security Review

Auditors

Guido Vranken, Lead Security Researcher

Blockdev, Security Researcher

Kyle Charbonnet, Security Researcher

Report prepared by: Lucas Goiriz

April 12, 2025

Contents

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

2 Introduction

Matter Labs is an engineering team passionate about liberty, blockchain, and math; also known as the inventors of ZKsync: Ethereum's most user-centric ZK rollup ZKsync.

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

Project Name	Matter Labs
Repository	zksync-protocol
Commit	4c44b9f8
Type of Project	DeFi, AMM
Audit Timeline	Mar 22nd to Apr 8th

Issues Found

Severity	Count	Fixed	Acknowledged
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
Total	**5**	**0**	**0**

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 == 44079209702962438423933672150061560849164694571458439784 61)
    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 == 16606911640375200216783280360720795247 8',
    'k1 > 166069116403752002167832803607207952478',
    'k2 == -160042871798160843020181221280528662475',
    'k2 < -16004287179816084302018122128052866 2475',
    '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 = -14794675688178931900076503080380341072 8,
 c1 = 0,
 g1 = 78266054408908085307878795501562853415 7,
 a2 = 14794675688178931901069635353818910849 1,
 c2 = 0,
 k2_lambda = 0,
 lambd = 44079209702962438423933672150061560849164694571458439784 61,
 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 = 78266054408908085307878795501562853415 7,
 q1 = -14693069900987479474151401527081222552986139269577563907 36,
 c1 = 9931322734385697762,
 a2 = 14794675688178931901069635353818910849 1,
 g2 = 52538187511802934231,
 lambd = 44079209702962438423933672150061560849164694571458439784 61,
 c2 = 14794675688178931900076503080380341072 6,
 b2 = 9931322734385697763,
 k2_lambda = 21888242871839275222246405745257275088252470886652455705666880156765044580872,
 k1 = 166069116403752002167832803607207952478,
 a1 = 9931322734385697763,
 n = 21888242871839275222246405745257275088548364400416034343698204186575808495617,
 k = 21888242871839275222246405745257275088418540003056207707834712960372252533350,
 q2 = -14693069900987479475630869095899115742795163122927884059 38,
 b1 = -14794675688178931900076503080380341072 8,
```

```
 k2 = -147946756881789318980902385335032015202]

Solving for k1 > 166069116403752002167832803607207952478
Solution does not exist

Solving for k2 == -160042871798160843020181221280528662475
Solution found:
[q1 = -14693069900987479472671933958263329362978488961539529800008,
 k2_lambda = 2188824287183927522022264089170920687196232042616713285287334616504426560617 4,
 a1 = 9931322734385697763,
 c1 = 9931322734385697761,
 n = 21888242871839275222246405745257275088548364400416034343698204186575808495617,
 k1 = 18122359521962683156272859136894859594,
 b1 = -147946756881789319000765030803803410728,
 c2 = 147946756881789318987086022921494283441,
 g1 = 78266054408908085307878795501562853415 7,
 k2 = -160042871798160843020181221280528662475,
 a2 = 14794675688178931901069635353818910849 1,
 k = 2188824287183927522022264089170920687198044278568909553602961902418116046576 8,
 g2 = 52538187511802934231,
 lambd = 4407920970296243842393367215006156084916469457145843978461,
 q2 = -14693069900987479474272362676244937793180301174344816424 83,
 b2 = 9931322734385697763]

Solving for k2 < -160042871798160843020181221280528662475
Solution does not exist

Solving for k2 == 4965661367192848883
Solution found:
[q1 = -44384027064536795700229509241141023218 4,
 b2 = 9931322734385697763,
 c1 = 3,
 k2_lambda = 66118814554443657635160774440683394678693560089496714134 46,
 a1 = 9931322734385697763,
 n = 21888242871839275222246405745257275088548364400416034343698204186575808495617,
 c2 = 44690952304735639927,
 k1 = 12840589556456651763209527287541672999 2,
 b1 = -147946756881789319000765030803803410728,
 g1 = 78266054408908085307878795501562853415 7,
 k2 = 4965661367192848883,
 a2 = 14794675688178931901069635353818910849 1,
 k = 66118814554443657636444833396329059855014512818250881434 38,
 q2 = -44384027064536795699732943104421738330 1,
 g2 = 52538187511802934231,
 lambd = 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:

- `k1` is never negative.
- Neither `abs(k1)` or `abs(k2)` 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*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": "8396832590500554279673173994121106164143113568083013035535311148239221659898 1",
+            "k1": "876657049807958515889393974150067471 13",
+            "k2": "5219985908569991869624972491476454823 1",
             "k2_negated": true
         },
         {
-            "k": "59681096324589447255856301372518141279002759342310841152722992182309039187 42",
-            "k1": "14836831004498481929843964842636346524 7",
```

```
-            "k1_negated": false,
-            "k2": "704367658756761062763074389597392023 43",
+            "k": "1157920892373161954235709850086879078532699846656405640394575840079131296399 35",
+            "k1": "958700012259823101000077815524245875 95",
+            "k2": "6399026299289356034422595942526016062 3",
             "k2_negated": true
         },
         {
-            "k": "1742498084139980620251305439875895103323009640850145693178755079904652572667",
-            "k1": "7939021705589158861357544018624332533 3",
-            "k1_negated": false,
-            "k2": "436919747965623403034435569717050072 73",
+            "k": "2188824287183927522224640574525727508841854000305620770783471296037225253335 0",
+            "k1": "1660691164037520021678328036072079524 78",
+            "k2": "1479467568817893189809023853350320152 02",
             "k2_negated": true
         },
         {
-            "k": "1243515812118320026814008448442902581429356161737069998329162229814064635303 9",
-            "k1": "5226346601997836651341157939180825969 0",
-            "k1_negated": false,
-            "k2": "1476215352996517202943635100878682103 46",
+            "k": "6611881455444365763644483339632905985501451281825088143438",
+            "k1": "1284058955645665176320952728754167299 92",
+            "k2": "4965661367192848883",
+            "k2_negated": false
+        },
+        {
+            "k": "2188824287183927522022264089170920687199858695806365282206259479620623118998 2",
+            "k1": "3626653189651996918924863116196558380 8",
+            "k2": "1600428717981608430201812212805286624 75",
+            "k2_negated": true
+        },
+        {
+            "k": "2188824287183927522224640574525727508854836440041603434366841021837265140232 8",
+            "k1": "1479467568817893189908337080694177129 65",
+            "k2": "1479467568817893189908337080694177129 65",
+            "k2_negated": true
+        },
+        {
+            "k": "1479467568817893190206276762725748062 54",
+            "k1": "9931322734385697763",
+            "k2": "9931322734385697763",
             "k2_negated": true
         },
         {
-            "k": "1733851479738228817254334653932863433958772029919540147158617787253437428141 2",
-            "k1": "6963752629513492780958781284008195957 0",
-            "k1_negated": false,
-            "k2": "1162420880821341453301122691017321611 98",
+            "k": "1157920892373161954235709850086879078531220379087587747204468876543749405314 45",
+            "k1": "9587000122598231010000778155242458759 6",
+            "k2": "6399026299289356033429463669087446286 0",
             "k2_negated": true
         },
         {
-            "k": "4424259225288216396589498511088720282107929032919356608555421459560833194920",
-            "k1": "9177151361611855993569459472141014672 2",
-            "k1_negated": false,
-            "k2": "5783929130781118479843327582549223339 3",
+            "k": "2188824287183927522224640574525727508841854000305620770783471296037225253335 1",
+            "k1": "1812235952196268315713645006901884398 8",
```

```
+            "k2": "14794675688178931899083370806941771296 5",
             "k2_negated": true
         },
         {
-            "k": "12703330644849743193798861201554125235863198764416081532769166352996028110428",
-            "k1": "70304310444027999311628774683115887163",
-            "k1_negated": false,
-            "k2": "18942669441150157207684220257508261454",
+            "k": "21888242871839275220222640891709206872012645081782417699208530339517524196747",
+            "k1": "50324655615284846325252851738872892810",
+            "k2": "12096114916371524019416190476725251747",
             "k2_negated": true
         },
         {
-            "k": "87796748895409960717354148130217387300930651012538188516419495459925788722 5",
-            "k1": "49277213207367892196602854559228360108",
-            "k1_negated": false,
-            "k2": "14679705262835984709727490630948317294 6",
+            "k": "11579208923731619542136702452353978593205515688576292871095710024730476692545 7",
+            "k1": "15169920729231968356310692628607841659 3",
+            "k2": "63990262992893560339260298058067311742",
             "k2_negated": true
         },
         {
-            "k": "10908826393692157466283479854939993281909445730438484613349035400366196560258",
-            "k1": "14419959583026184288919390290764209063 5",
-            "k1_negated": false,
-            "k2": "15261296491077073016255383100821336547 1",
+            "k": "11579208923731619541695910355324354208960596046469043525257205734154652693486 5",
+            "k1": "95870001225982310094533477673682404462",
+            "k2": "63990262992893560339260298058067311743",
+            "k2_negated": true
+        },
+        {
+            "k": "21888242871839275220222640891709206871980442785689095536029619024181160465768",
+            "k1": "18122359521962683156272859136894859594",
+            "k2": "16004287179816084302018122128052866247 5",
+            "k2_negated": true
+        },
+        {
+            "k": "21888242871839275222246405745257275088418540003056207707834712960372252533351",
+            "k1": "18122359521962683157136450069018843988",
+            "k2": "14794675688178931899083370806941771296 5",
+            "k2_negated": true
+        },
+        {
+            "k": "15841190656464805753193627983676170750939600321456310067 0",
+            "k1": "10633823966279326983230456482242756609",
+            "k2": "10633823966279326994231158980716092773",
+            "k2_negated": true
+        },
+        {
+            "k": "31682381312929611491592580279173409600809565289093709284 8",
+            "k1": "21267647932558653966460912964485513217",
+            "k2": "21267647932558653978530995227046487783",
+            "k2_negated": true
+        },
+        {
+            "k": "50415685965548360719293253068260456389112680558513399471 3",
+            "k1": "42535295865117307932921825928971026432",
+            "k2": "42535295865117307937199344985321580039",
+            "k2_negated": true
```

```
+        },
+        {
+            "k": "787917670796155022258799362770811857585895320636066335087O",
+            "k1": "85070591730234615865843651857942052864",
+            "k2": "85070591730234615869433028603450311197",
+            "k2_negated": true
+        },
+        {
+            "k": "85070591730234615865843651857942052863",
+            "k1": "85070591730234615865843651857942052863",
+            "k2": "0",
+            "k2_negated": false
+        },
+        {
+            "k": "158216890268195359853818065135426732989046920237836929401825353232944797927 72",
+            "k1": "13099559360135235052400423830238750792",
+            "k2": "85070591730234615865843651857942052863",
             "k2_negated": true
         },
         {
-            "k": "187193344888606275003699578234285392125555365333613156557163335924320860884 05",
-            "k1": "78693297012568138413531716443444140114",
-            "k1_negated": false,
-            "k2": "84127714618519115178173319957441990163",
+            "k": "158216890268195359853818065135426732989766630561537923209959785513221830948 43",
+            "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": "69287658908343637987657105354283899753338979007127849066532821101257861420 62"
+                "x": "1",
+                "y": "2"
             },
-            "scalar": "13650076562025738285589406854928154499107354233219361696036823113035450875054",
+            "scalar": "83968325905005542796731739941211061641431135680830130355353111482392216598981",
             "expected": {
-                "x": "11299373567935086735078551232826217925502745784801646334636571278818215743539",
-                "y": "18836805603793619172102959251973968032345476967462197112983508530862248992053"
+                "x": "13925104235512444333709620682324960539413781169336437097084520151760295826 5",
+                "y": "43206242299667804525193550069936797789175867887206816142719245715702923082 16"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "11579208923731619542357098500868790785326998466564056403945758400791312963 9935",
+            "expected": {
+                "x": "21415159568991615317144600033915305503576371596506956373206836402282692989 778",
+                "y": "85730708963198648685359335622646230764206529263032379820786930681476572432 87"
```

```
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522224640574525727508841854000305620770783471296037225253335O",
+            "expected": {
+                "x": "57933461905690075466884301347726452314736107517139451824874695081203300773",
+                "y": "13543497425240060327382100399841884843806450700078130961320280218982647557466"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "661188145544436576364448333963290598550145128182508814343O",
+            "expected": {
+                "x": "178437960941538044754754620080694899677464359212977570215831660151025775310",
+                "y": "1408787244828519168627565136031249644639553350965615731423854082057517671O3"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522022264089170920687199858695806365282206259479620623118998O",
+            "expected": {
+                "x": "12787867861020980678117956101723515349350391304192238540091144947347208088030",
+                "y": "13240302848311906527618553598894808278134638473061458549401574734857896047383"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522224640574525727508854836440041603434366841021837265140232O",
+            "expected": {
+                "x": "13194957371956931795709136998963250104760578165429223396476574392416761212575",
+                "y": "10065223513748416994106058133200810928700988293106150653510602746184807367191"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "14794675688178931902062767627257480625O",
+            "expected": {
+                "x": "153614446392720005420797510841675154134008307965343626901032457091530057416O4",
+                "y": "18893506883835740211889523270233281666885803481890970229437899355752901794202"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+
```

```
+            "scalar": "1157920892373161954235709850086879078531220379087587747204468876543749405314445",
+            "expected": {
+                "x": "703836339723672555650303490191661487819825927362584777124527837672382346800 4",
+                "y": "112042639532234258865662927446552625916177324237961851208732174399827883304 90"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522224640574525727508841854000305620770783471296037225253335 1",
+            "expected": {
+                "x": "106382220095516501623683998629890832998015159405056087723545553858502864391 55",
+                "y": "202645297636873160345502878229910090828979708059695108809745350010061012577 11"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522022264089170920687201264508178241769920853033951752419674 7",
+            "expected": {
+                "x": "914642297118378675675606111197031621150361197937088141398366580945006317729 5",
+                "y": "212083223451218208582114044341660588767842914317201587140232983306712806689 09"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "1157920892373161954213670245235397859320551568857629287109571002473047669254 57",
+            "expected": {
+                "x": "136186461543677497966829064064586885113154738030057999131511511920284190802 22",
+                "y": "113477015922621666976326600449168420966976339713856288414288486948973806217 66"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "1157920892373161954169591035532435420896059604646904352525720573415465269348 65",
+            "expected": {
+                "x": "109379235876867632305638753833596882954544261110301332068691995070502041063 36",
+                "y": "213781068482073442504924259512667150080955518584319847310627307485079052122 4"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522022264089170920687198044278568909553602961902418116046576 8",
+            "expected": {
+                "x": "185426741796420489225136113224675139053682813856958291340817620173127582617 85",
+                "y": "154712407290953541973266635062013968918287647202828221935020765009977557011 45"
+            }
+        },
+        {
```

```
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "2188824287183927522224640574525727508841854000305620770783471296037225253335 1",
+            "expected": {
+                "x": "106382220095516501623683998629890832998015159405056087723545553858502864391 55",
+                "y": "202645297636873160345502878229910090828979708059695108809745350010061012577 11"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "158411906564648057531936279836761707509396003214563100670",
+            "expected": {
+                "x": "194718462760420196036803075845442743259118460661630103746653433049818358970 92",
+                "y": "182771122071027074289497915406337041691663842327919438471578017888084099245 10"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "3168238131292961149159258027917340960080956528909370928 48",
+            "expected": {
+                "x": "861560855474980543379328681622940873093930894954072904007939373277023003172",
+                "y": "131314115895916044475604585989035223199968736993571207598583082755800144616 36"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "504156859655483607192932530682604563891126805585133994717 3",
+            "expected": {
+                "x": "115555982361059800901625897411890897578358490696362866650406924813443353972 66",
+                "y": "166409429333586027870221042414476551902201918330329095807402714149778659906 16"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "787917670796155022258799362770811857585895320636066335087 0",
+            "expected": {
+                "x": "144403137143233231408807354125994764590694516069501061078345417836492432120 96",
+                "y": "171303125101377484471077390097268801031418995439363935034611925410315036313 47"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "850705917302346158658436518579420528 63",
+            "expected": {
+                "x": "110771784344114453298237121343448071815181522988711089545386122621759041140 84",
```

```
+                "y": "13585835793919331183767088957122633361381220469170285909154452889609259908 6"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "158216890268195359853818065135426732989046920237836929401825353232944797927 72",
+            "expected": {
+                "x": "484164496523022757342195536722941604415763830701845898364236307015290437519 3",
+                "y": "162613529506994064978761618003968819207462436319025843509088512666734995077 45"
+            }
+        },
+        {
+            "point": {
+                "x": "1",
+                "y": "2"
+            },
+            "scalar": "158216890268195359853818065135426732989766630561537923209959785513221830948 43",
+            "expected": {
+                "x": "900040308848938488200802095870749752869804549622465977901137056534900166692 8",
+                "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 = 0x24ccef014a773d2cf7a7bd9d4391eb18d
g2 = 0x2d91d232ec7e0b3d7
lambd = 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 * lambd % 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": 218882428718392752222464057452572750884185400030562077078347129603722525333350,
         "k1": 166069116403752002167832803607207952478,
         "k2": -147946756881789318980902385335032015202,
        },
        # Largest possible value for k2 (with 0 < k < n)
        {
         'k' : 661188145544436576364448333963290598550145128182508814343 8,
         'k1' : 128405895564566517632095272875416729992,
         'k2' : 4965661367192848883,
        },
        # Smallest possible value for k2 (with 0 < k < n)
        {
         'k': 218882428718392752202226408917092068719985869580636528220625947962062311899 82,
         'k1': 362665318965199691892486311619655838 08,
         'k2': -160042871798160843020181221280528662475,
        },
        # abs(k1) == abs(k2) (with 0 < k < n)
        {
         'k': 218882428718392752222464057452572750885483644004160343436684102183726514023 28,
         'k1': 147946756881789318990833708069417712965,
         'k2': -147946756881789318990833708069417712965,
        },
        # Smallest k with abs(k1) == abs(k2) (with 0 < k < n)
        {
         "k": 147946756881789319020627676272574806254,
         "k1": 9931322734385697763,
         "k2": -9931322734385697763,
        },
        # Smallest possible q2 (-777285445481856841838900206857070116356345085816693829 3028) with 0 < k
        ↪  < 2**256
        {
         "k": 115792089237316195423570985008687907853122037908758774720446887654374940531445,
         "k1": 109441214359196376111232028726286375442837692003306154028600960037165852763444,
         "k2": -211937019874682879335059667494677873588,
        },
```

```
# Smallest possible q2 (-14693069900987479475630869095899115742894476350271741037O1) with 0 < k
↪  < n
{
 "k": 2188824287183927522224640574525727508841854000305620770783471296037225253335 1,
 "k1": 1812235952196268315713645006901884398 8,
 "k2": -14794675688178931899083370806941771296 5,
},
# Smallest possible q1 (-14693069900987479474151401527081222552986139269577563907 36) with 0 < k
↪  < n
{
 "k": 2188824287183927522022264089170920687201264508178241769920853033951752419674 7,
 "k1": 5032465561528484632525285173887289281 0,
 "k2": -1209611491637152401941619047672525174 7,
},
# Smallest possible q1 (-77728544548185684181770650486960182842283911906722604194 40) with 0 < k
↪  < 2**256
{
 "k": 1157920892373161954213670245235397859320551568857629287109571002473047669254 57,
 "k1": 1094412143591963761112320287262863754428935212093724914020541278591651208946 78,
 "k2": -6399026299289356033926029805806731174 2,
},
# Largest difference between q1 and q2 (2119370198746828793400253288618707224 71) with 0 < k <
↪  2**256
{
 "k": 1157920892373161954169591035532435420896059604646904352525720573415465269348 65,
 "k1": 1094412143591963761112320287262863754428376920033061540285954857332871105803 10,
 "k2": -2119370198746828793400253288618707224 71,
},
# Largest difference between q1 and q2 (1600428717981608430201812212805286624 75) with 0 < k < n
{
 "k": 2188824287183927522022264089170920687198044278568909553602961902418116046576 8,
 "k1": 1812235952196268315627285913689485959 4,
 "k2": -1600428717981608430201812212805286624 75,
},
# Largest k2_lambda
↪  (2188824287183927522224640574525727508840041764353424502467757651030323368936 3) with 0 < k <
↪  n
{
 "k": 2188824287183927522224640574525727508841854000305620770783471296037225253335 1,
 "k1": 1812235952196268315713645006901884398 8,
 "k2": -14794675688178931899083370806941771296 5,
},
# Large k1, k2 (>= 2**123)
{
 "k": 1584119065646480575319362798367617075093960032145631006 70,
 "k1": 1063382396627932698323045648224275660 9,
 "k2": -1063382396627932699423115898071609277 3,
},
# Large k1, k2 (>= 2**124)
{
 "k": 3168238131292961149159258027917340960080956528909370928 48,
 "k1": 2126764793255865396646091296448551321 7,
 "k2": -2126764793255865397853099522704648778 3,
},
# Large k1, k2 (>= 2**125)
{
 "k": 5041568596554836071929325306826045638911268055851339947173,
 "k1": 4253529586511730793292182592897102643 2,
 "k2": -4253529586511730793719934498532158003 9,
},
# Large k1, k2 (>= 2**126)
{
```

```
        "k": 787917670796155022258799362770811857585895320636066335087O,
        "k1": 85070591730234615865843651857942052864,
        "k2": -85070591730234615869433028603450311197,
       },
       # k1 with largest popcount (126)
       {
        "k": 85070591730234615865843651857942052863,
        "k1": 85070591730234615865843651857942052863,
        "k2": 0,
       },
       # k2 with largest popcount (126)
       {
        "k": 158216890268195359853818065135426732989046920237836929401825353232944797927 72,
        "k1": 13099559360135235052400423830238750792,
        "k2": -85070591730234615865843651857942052863,
       },
       # k1, k2 with largest popcount (126+126)
       {
        "k": 158216890268195359853818065135426732989766630561537923209959785513221830948 43,
        "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 * lambd) % n == k % n)

write_decomposition_tests()
write_ecmul_tests()
```

This is a 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:
        # λ = (y - y)/(x - x)
        lambda_1 = (y2-y1)/(x2-x1)

        # x = λ(λ + a) - a - x - x
        x3 = lambda_1*(lambda_1+a1) - a2 - x1 - x2

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

        # x = λ(λ + a) - a - x - x
        new_x_canonical = lambda_2*(lambda_2+a1) - a2 - x1 - x3
        # y = λ(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. Nonethless, my suggestion would be to either implement complete point tranformation 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: `k2` here is calculated as `q2 - q1` after negating `q1`. `k2` can be calculated as `q2 + q1` if `q1` 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 multi-pairing. 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(...)`.