CANTINA

Ethereum Pectra: Ethereum Protocol

Competition

July 21, 2025

Contents

1 Introduction

1.1 About Cantina

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

1.2 Disclaimer

A competition provides a broad evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While competitions endeavor to identify and disclose all potential security issues, they cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities, therefore, any changes made to the code would require an additional security review. Please be advised that competitions are not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
High	*Must* fix as soon as possible (if already deployed) and can be triggered by any user without significant constraints, generating outsized returns to the exploiter. For example: loss of user funds (significant amount of funds being stolen or lost) or breaking core functionality (failure in fundamental protocol operations).
Medium	Global losses <10% or losses to only a subset of users, requiring significant constraints (capital, planning, other users...) to be exploited. For example: temporary disruption or denial of service (DoS), minor fund loss or exposure or breaking non-core functionality
Low	Losses will be annoying but easily recoverable, requiring unusual scenarios or admin actions to be exploited.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above matrix. High severity findings represent the most critical issues that must be addressed immediately, as they either have high impact and high likelihood of occurrence, or medium impact with high likelihood.

Medium severity findings represent issues that, while not immediately critical, still pose significant risks and should be addressed promptly. These typically involve scenarios with medium impact and medium likelihood, or high impact with low likelihood.

Low severity findings represent issues that, while not posing immediate threats, could potentially cause problems in specific scenarios. These typically involve medium impact with low likelihood, or low impact with medium likelihood.

Lastly, some findings might represent improvements that don't directly impact security but could enhance the codebase's quality, readability, or efficiency (Gas and Informational findings).

2 Security Review Summary

Ethereum is a worldwide system, an open-source platform to write computer code that stores and automates digital databases using smart contracts, without relying upon a central intermediary, solving trust with cryptographic techniques.

From Feb 21st to Mar 27th Cantina hosted a competition based on the Ethereum Pectra upgrade. The present report focuses in the None implementation. The participants identified a total of **4** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 0
- Low Risk: 0
- Gas Optimizations: 0
- Informational: 3

3 Findings

3.1 High Risk

3.1.1 Ambiguous Point at Infinity Handling in EIP-2537 Could Lead to Network Split and Mass Validator Slashing

Submitted by NDKoo

Severity: High Risk

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

Summary: EIP-2537's ambiguous specification regarding point at infinity handling in BLS pairing checks could lead to inconsistent implementations across Ethereum clients. This inconsistency could result in network splits and potential mass validator slashing due to different validation behaviors.

Finding Description: EIP-2537 states:

> If any input is the infinity point, pairing result will be 1. Protocols may want to check and reject infinity points prior to calling the precompile.

This specification is problematic for several reasons:

- Mathematical Ambiguity: For a single pairing $e(P, Q)$, if either P or Q is infinity, then $e(P, Q) = 1$. For multiple pairings $\prod_i e(P_i, Q_i) = 1$, the specification doesn't clearly define how infinity points should affect the final result. The statement "*pairing result will be 1*" is unclear whether it applies to individual pairings or the final product.
- Implementation Divergence: Different clients have interpreted this ambiguity differently:
  - (`execution-specs/src/ethereum/prague/vm/precompiled_contracts/bls12_381/bls12_381_pairing.py`, Ethereum-Protocol spec):

```
    result = FQ12.one()
        for i in range(k):
            g1_start = Uint(384 * i)
            g2_start = Uint(384 * i + 128)

            g1_point = bytes_to_G1(buffer_read(data, U256(g1_start), U256(128)))
            if multiply(g1_point, curve_order) is not None:
                raise InvalidParameter("Sub-group check failed.")

            g2_point = bytes_to_G2(buffer_read(data, U256(g2_start), U256(256)))
            if multiply(g2_point, curve_order) is not None:
                raise InvalidParameter("Sub-group check failed.")

            result *= pairing(g2_point, g1_point)

        if result == FQ12.one():
            evm.output = b"\x00" * 31 + b"\x01"
        else:
            evm.output = b"\x00" * 32
```

  - (`src/Nethermind/Nethermind.Evm/Precompiles/Bls/PairingCheckPrecompile.cs`, Nethermind):

```
    bool hasInf = false;
        for (int i = 0; i < inputData.Length / PairSize; i++)
        {
            int offset = i * PairSize;

            if (!x.TryDecodeRaw(inputData[offset..(offset + BlsConst.LenG1)].Span) ||
                !(BlsConst.DisableSubgroupChecks || x.InGroup()) ||
                !y.TryDecodeRaw(inputData[(offset + BlsConst.LenG1)..(offset + PairSize)].Span) ||
                !(BlsConst.DisableSubgroupChecks || y.InGroup()))
            {
                return IPrecompile.Failure;
            }


            // x == inf || y == inf -> e(x, y) = 1
            if (x.IsInf() || y.IsInf())
            {
                hasInf = true;
                continue;
            }

            p.MillerLoop(y, x);
            acc.Mul(p);
        }

        bool verified = hasInf || acc.FinalExp().IsOne();
        byte[] res = new byte[32];
        if (verified)
        {
            res[31] = 1;
        }

        return (res, true);
```

These implementations show fundamentally different approaches to handling infinity points.

Impact Explanation:

- Cryptographic Impact: BLS signature verification relies on correct pairing validation. Invalid signature combinations could be accepted by Nethermind but rejected by other clients. This breaks the security properties of BLS signatures.
- Consensus Impact: Different behavior across clients could lead to consensus failures. This becomes critical when BLS signatures are used in consensus-critical protocols.
- Protocol Security: Protocols relying on BLS signature verification might be compromised. The vulnerability could be exploited to create valid-looking but mathematically incorrect signatures.

Likelihood Explanation:

- Reproducibility: The issue is deterministic and can be consistently reproduced. Test cases demonstrate the divergent behavior between implementations.
- Accessibility: The precompile is publicly accessible. No special conditions or permissions are needed to trigger the behavior.
- Complexity: The attack vector is relatively simple. It only requires crafting inputs with specific infinity points.

Proof of Concept: Test cases demonstrate the behavior:

- `data1`: Valid pairing - all clients accept.
- `data2`, `data3`: Invalid pairings with infinity points - Nethermind accepts, others rejects.

Nethermind test: Replace the proof of concept with `src/Nethermind/Nethermind.Evm.Test/BlsPairingCheckPrecompileTests.cs` and run the command:

```
# cd nethermind/src/Nethermind:
# dotnet test Nethermind.Evm.Test/Nethermind.Evm.Test.csproj --filter
↪  "FullyQualifiedName=Nethermind.Evm.Test.BlsPairingCheckPrecompileTests" -c release --verbosity detailed
```

```
// SPDX-FileCopyrightText: 2022 Demerzel Solutions Limited
// SPDX-License-Identifier: LGPL-3.0-only
```

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using FluentAssertions;
using Nethermind.Core.Extensions;
using Nethermind.Evm.Precompiles;
using Nethermind.Evm.Precompiles.Bls;
using Nethermind.Specs.Forks;
using NUnit.Framework;

namespace Nethermind.Evm.Test;

public class BlsPairingCheckPrecompileTests
{
    [Test]
    public void Test()
    {
        foreach ((byte[] input, ReadOnlyMemory<byte> expectedResult) in Inputs)
        {
            Console.WriteLine($"Testing input: {input.ToHexString()}");
            Debug.WriteLine($"Expected result: {expectedResult.ToArray().ToHexString()}");

            IPrecompile precompile = PairingCheckPrecompile.Instance;
            (ReadOnlyMemory<byte> output, bool success) = precompile.Run(input, MuirGlacier.Instance);

            TestContext.Out.WriteLine($"Output: {output.ToArray().ToHexString()}");
            TestContext.Out.WriteLine($"Success: {success}");

            output.ToArray().Should().BeEquivalentTo(expectedResult.ToArray());
            success.Should().BeTrue();
        }
    }

    /// <summary>
    /// https://github.com/matter-labs/eip1962/tree/master/src/test/test_vectors/eip2537
    /// </summary>
    private static readonly Dictionary<byte[], ReadOnlyMemory<byte>> Inputs = new()
    {
        // Test case 1: Single pair with G1 point at infinity
        { Bytes.FromHexString(
            // G1 point
            "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"+
            "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"+
            // G2 point
            "00000000000000000000000000000000017c9fcf0504e62d3553b2f089b64574150aa5117bd3d2e89a8c1ed59bb7f70fb8"+
            "3215975ef31976e757abf60a75a1d9f"+
            "00000000000000000000000000000000008f5a53d704298fe0cfc955e020442874fe87d5c729c7126abbdcbed355eef6c8"+
            "f07277bee6d49d56c4ebaf334848624"+
            "0000000000000000000000000000000001302dcc50c6ce4c28086f8e1b43f9f65543cf598be440123816765ab6bc93f62b"+
            "ceda80045fbcad8598d4f32d03ee8fa"+
            "00000000000000000000000000000000bbb4eb37628d60b035a3e0c45c0ea8c4abef5a6ddc5625e0560097ef9caab208"+
            "221062e81cd77ef72162923a1906a40"
            ),
            Bytes.FromHexString("0000000000000000000000000000000000000000000000000000000000000001")
        },

        // Test case 2: Multiple pairs with G1 point at infinity
        // This test case should fail because e(G1-1,G2-1) != e(G1-2,G2-2)
        // origin G1-1 : 00000000000000000000000000000000001830f52d9bff64a623c6f5259e2cd2c2a08ea17a8797aaf83174e
        //  a1e8c3bd3955c2af1d39bfa474815bfe60714b7cd80
        //                00000000000000000000000000000000874389c02d4cf1c61bc54c4c24def11dfbe7880bc998a95e700
        //  63009451ee8226fec4b278aade3a7cea55659459f1d5
        { Bytes.FromHexString(
            // G1 point 1 (modified)
            "00000000000000000000000000000000012196c5a43d69224d8713389285f26b98f86ee910ab3dd668e413738282003cc5"+
            "b7357af9a7af54bb713d62255e80f56" +
            "0000000000000000000000000000000006ba8102bfbeea4416b710c73e8cce3032c31c6269c44906f8ac4f7874ce99fb1"+
            "755992486528963884ce429a992fee" +
            // G2 point 1
            "00000000000000000000000000000000197737f831d4dc7e708475f4ca7ca15284db2f3751fcaac0c17f517f1ddab35e1"+
            "a37907d7b99b39d6c8d9001cd50e79e" +
```

```
    "00000000000000000000000000000000af1a3f6396f0c983e7c2d42d489a3ae5a3ff0a553d93154f73ac770cd0af7467
    ↪  aa0cef79f10bbd34621b3ec9583a834" +
    "00000000000000000000000000000000001918cb6e448ed69fb906145de3f11455ee0359d030e90d673ce050a360d796de3
    ↪  3ccd6a941c49a1414aca1c26f9e699e" +
    "0000000000000000000000000000000019a915154a13249d784093facc44520e7f3a18410ab2a3093e0b12657788e9419
    ↪  eec25729944f7945e732104939e7a9e" +
    // G1 point 2 (modified)
    "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    ↪  00000000000000000000000000000000" +
    "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    ↪  00000000000000000000000000000000" +
    // G2 point 2
    "00000000000000000000000000000000197737f831d4dc7e708475f4ca7ca15284db2f3751fcaac0c17f517f1ddab35e1
    ↪  a37907d7b99b39d6c8d9001cd50e79e" +
    "00000000000000000000000000000000af1a3f6396f0c983e7c2d42d489a3ae5a3ff0a553d93154f73ac770cd0af7467
    ↪  aa0cef79f10bbd34621b3ec9583a834" +
    "00000000000000000000000000000000001918cb6e448ed69fb906145de3f11455ee0359d030e90d673ce050a360d796de3
    ↪  3ccd6a941c49a1414aca1c26f9e699e" +
    "0000000000000000000000000000000019a915154a13249d784093facc44520e7f3a18410ab2a3093e0b12657788e9419
    ↪  eec25729944f7945e732104939e7a9e"
    ),
    Bytes.FromHexString("0000000000000000000000000000000000000000000000000000000000000001")
},

// Test case 3: Multiple pairs with G2 point at infinity
// This test case should fail because e(G1-1,G2-1) != e(G1-2,G2-2)
// correct G1-1 : 00000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adee
↪  d713bae79d606041406df26169c35df63cf972c94be1
//                00000000000000000000000000000011bc8afe71676e6730702a46ef817060249cd06cd82e69810850
↪  12ff6d013aa4470ba3a2c71e13ef653e1e223d1ccfe9
{ Bytes.FromHexString(
    // G1 point 1 (modified)
    "00000000000000000000000000000012196c5a43d69224d8713389285f26b98f86ee910ab3dd668e413738282003cc5
    ↪  b7357af9a7af54bb713d62255e80f56" +
    "00000000000000000000000000000006ba8102bfbeea4416b710c73e8cce3032c31c6269c44906f8ac4f7874ce99fb1
    ↪  7559992486528963884ce429a992fee" +
    // G2 point 1
    "00000000000000000000000000000013a3de1d25380c44ca06321151e89ca22210926c1cd4e3c1a9c3aa6c709ab5fdd
    ↪  00f8df19243ce058bc753ccf03424ed" +
    "00000000000000000000000000001657dbebf712cbda6f15d1d387c87b3fb9b386d5d754135049728a2a856ba2944
    ↪  c741024131a93c78655fdb7bfe3c803" +
    "00000000000000000000000000000068edef3169c58920509ed4e7069229bd8038a45d2ce5773451cc18b396d2838c
    ↪  9539ecb52298a27eebd714afacb907c" +
    "00000000000000000000000000000004c5346765a62f2d2e700aadccf747acb3322c250435ce2cf358c08f1e286427c
    ↪  abace052327c4b30135c8482c5c0eb9" +
    // G1 point 2
    "00000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adeed713bae79d606
    ↪  041406df26169c35df63cf972c94be1" +
    "00000000000000000000000000000084486ebc81878331aab7d6f53ca3c773fda7b181b56a93e5ee0bfa189afbb7fd
    ↪  7a05c5bea35ec1054c0e1ddc2e2dac2" +
    // G2 point 2 (modified)
    "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    ↪  00000000000000000000000000000000" +
    "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    ↪  00000000000000000000000000000000" +
    "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    ↪  00000000000000000000000000000000" +
    "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    ↪  00000000000000000000000000000000" +
    // G1 point 3
    "00000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adeed713bae79d606
    ↪  041406df26169c35df63cf972c94be1" +
    "00000000000000000000000000000011bc8afe71676e6730702a46ef817060249cd06cd82e6981085012ff6d013aa44
    ↪  70ba3a2c71e13ef653e1e223d1ccfe9" +
    // G2 point 3
    "00000000000000000000000000000013a3de1d25380c44ca06321151e89ca22210926c1cd4e3c1a9c3aa6c709ab5fdd
    ↪  00f8df19243ce058bc753ccf03424ed" +
    "00000000000000000000000000001657dbebf712cbda6f15d1d387c87b3fb9b386d5d754135049728a2a856ba2944
    ↪  c741024131a93c78655fdb7bfe3c803" +
    "00000000000000000000000000000137232f722e38e084611ba67d2e28a3b8c73c13f20b6bb4c22141115bd43cdeb5
    ↪  55861335f2a75d7cb418eb505341a2f" +
```

```
            "00000000000000000000000000000000153bdd82d3d9b76d1cab9d087654652ab1451f5fef4f449273d81211d88891fc5
            ↪  3f131f98e2c3b4cb8c937b7d3a39bf2" +
            // G1 point 4
            "00000000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adeed713bae79d606
            ↪  041406df26169c35df63cf972c94be1" +
            "00000000000000000000000000000000084486ebc81878331aab7d6f53ca3c773fda7b181b56a93e5ee0bfa189afbb7fd
            ↪  7a05c5bea35ec1054c0e1ddc2e2dac2" +
            // G2 point 4
            "00000000000000000000000000000000013a3de1d25380c44ca06321151e89ca22210926c1cd4e3c1a9c3aa6c709ab5fdd
            ↪  00f8df19243ce058bc753ccf03424ed" +
            "00000000000000000000000000000000001657dbebf712cbda6f15d1d387c87b3fb9b386d5d754135049728a2a856ba2944
            ↪  c741024131a93c78655fdb7bfe3c803" +
            "00000000000000000000000000000000137232f722e38e084611ba67d2e28a3b8c73c13f20b6bb4c22141115bd43cdeb5
            ↪  55861335f2a75d7cb418eb505341a2f" +
            "00000000000000000000000000000000153bdd82d3d9b76d1cab9d087654652ab1451f5fef4f449273d81211d88891fc5
            ↪  3f131f98e2c3b4cb8c937b7d3a39bf2"
            ),
            Bytes.FromHexString("0000000000000000000000000000000000000000000000000000000000000001")
        },
    };
}
```

Ethereum Protocol Spec: Create proof of concept file `execution-specs/src/ethereum/prague/vm/precompiled_contracts/bls12_381/bls12_381_test.py` and run the following command:

```
# cd ethereum-protocol/execution-specs/src/ethereum/prague/vm/precompiled_contracts/bls12_381
python bls12_381_test.py
```

```
from py_ecc.bls12_381.bls12_381_curve import FQ12, curve_order, multiply
from py_ecc.bls12_381.bls12_381_pairing import pairing
from ethereum.utils.byte import right_pad_zero_bytes
from ethereum_types.numeric import U256, Uint
from ethereum_types.bytes import Bytes

import sys
import os

current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(os.path.dirname(c
↪  urrent_dir))))))
sys.path.append(parent_dir)

from src.ethereum.prague.vm.precompiled_contracts.bls12_381 import bytes_to_G1, bytes_to_G2

def buffer_read(buffer: Bytes, start_position: U256, size: U256) -> Bytes:
    """
    Read bytes from a buffer. Padding with zeros if necessary.

    Parameters
    ----------
    buffer :
        Memory contents of the EVM.
    start_position :
        Starting pointer to the memory.
    size :
        Size of the data that needs to be read from `start_position`.

    Returns
    -------
    data_bytes :
        Data read from memory.
    """
    return right_pad_zero_bytes(
        buffer[start_position : Uint(start_position) + Uint(size)], size
    )

if __name__ == "__main__":
    def test_pairing(data):
        if len(data) == 0 or len(data) % 384 != 0:
            raise InvalidParameter("Invalid Input Length")

        k = len(data) // 384
        result = FQ12.one()
        for i in range(k):
```

```
        g1_start = Uint(384 * i)
        g2_start = Uint(384 * i + 128)
        g1_point = bytes_to_G1(buffer_read(data, U256(g1_start), U256(128)))
        g2_point = bytes_to_G2(buffer_read(data, U256(g2_start), U256(256)))
        result *= pairing(g2_point, g1_point)
    if result == FQ12.one():
        print("1")
    else:
        print("0")

data1 = bytes.fromhex(
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        ↪  "0000000000000000000000000000000000000000000000000000000000000000"+
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        ↪  "0000000000000000000000000000000000000000000000000000000000000000"+
        "0000000000000000000000000000000017c9fcf0504e62d3553b2f089b64574150aa5117bd3d2e89a8c1ed59bb7f70fb8"
        ↪  "3215975ef31976e757abf60a75a1d9f"+
        "0000000000000000000000000000000008f5a53d704298fe0cfc955e020442874fe87d5c729c7126abbdcbed355eef6c8"
        ↪  "f07277bee6d49d56c4ebaf334848624"+
        "0000000000000000000000000000000001302dcc50c6ce4c28086f8e1b43f9f65543cf598be440123816765ab6bc93f62b"
        ↪  "ceda80045fbcad8598d4f32d03ee8fa"+
        "0000000000000000000000000000000000bbb4eb37628d60b035a3e0c45c0ea8c4abef5a6ddc5625e0560097ef9caab208"
        ↪  "221062e81cd77ef72162923a1906a40"
    )

data2 = bytes.fromhex(
        "0000000000000000000000000000000012196c5a43d69224d8713389285f26b98f86ee910ab3dd668e413738282003cc5"
        ↪  "b7357af9a7af54bb713d62255e80f56" +
        "0000000000000000000000000000000006ba8102bfbeea4416b710c73e8cce3032c31c6269c44906f8ac4f7874ce99fb1"
        ↪  "755992486528963884ce429a992fee" +
        "0000000000000000000000000000000197737f831d4dc7e708475f4ca7ca15284db2f3751fcaac0c17f517f1ddab35e1"
        ↪  "a37907d7b99b39d6c8d9001cd50e79e" +
        "0000000000000000000000000000000af1a3f6396f0c983e7c2d42d489a3ae5a3ff0a553d93154f73ac770cd0af7467"
        ↪  "aa0cef79f10bbd34621b3ec9583a834" +
        "0000000000000000000000000000000001918cb6e448ed69fb906145de3f11455ee0359d030e90d673ce050a360d796de3"
        ↪  "3ccd6a941c49a1414aca1c26f9e699e" +
        "0000000000000000000000000000000019a915154a13249d784093facc44520e7f3a18410ab2a3093e0b12657788e9419"
        ↪  "eec25729944f7945e732104939e7a9e" +
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        ↪  "0000000000000000000000000000000000000000000000000000000000000000" +
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        ↪  "0000000000000000000000000000000000000000000000000000000000000000" +
        "0000000000000000000000000000000197737f831d4dc7e708475f4ca7ca15284db2f3751fcaac0c17f517f1ddab35e1"
        ↪  "a37907d7b99b39d6c8d9001cd50e79e" +
        "0000000000000000000000000000000af1a3f6396f0c983e7c2d42d489a3ae5a3ff0a553d93154f73ac770cd0af7467"
        ↪  "aa0cef79f10bbd34621b3ec9583a834" +
        "0000000000000000000000000000000001918cb6e448ed69fb906145de3f11455ee0359d030e90d673ce050a360d796de3"
        ↪  "3ccd6a941c49a1414aca1c26f9e699e" +
        "0000000000000000000000000000000019a915154a13249d784093facc44520e7f3a18410ab2a3093e0b12657788e9419"
        ↪  "eec25729944f7945e732104939e7a9e"
    )

data3 = bytes.fromhex(
        "0000000000000000000000000000000012196c5a43d69224d8713389285f26b98f86ee910ab3dd668e413738282003cc5"
        ↪  "b7357af9a7af54bb713d62255e80f56" +
        "0000000000000000000000000000000006ba8102bfbeea4416b710c73e8cce3032c31c6269c44906f8ac4f7874ce99fb1"
        ↪  "755992486528963884ce429a992fee" +
        "0000000000000000000000000000000013a3de1d25380c44ca06321151e89ca22210926c1cd4e3c1a9c3aa6c709ab5fdd"
        ↪  "00f8df19243ce058bc753ccf03424ed" +
        "0000000000000000000000000000000001657dbebf712cbda6f15d1d387c87b3fb9b386d5d754135049728a2a856ba2944"
        ↪  "c741024131a93c78655fdb7bfe3c803" +
        "000000000000000000000000000000000068edef3169c58920509ed4e7069229bd8038a45d2ce5773451cc18b396d2838c"
        ↪  "9539ecb52298a27eebd714afacb907c" +
        "0000000000000000000000000000000004c5346765a62f2d2e700aadccf747acb3322c250435ce2cf358c08f1e286427c"
        ↪  "abace052327c4b30135c8482c5c0eb9" +
        "0000000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adeed713bae79d606"
        ↪  "041406df26169c35df63cf972c94be1" +
        "0000000000000000000000000000000084486ebc81878331aab7d6f53ca3c773fda7b181b56a93e5ee0bfa189afbb7fd"
        ↪  "7a05c5bea35ec1054c0e1ddc2e2dac2" +
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        ↪  "0000000000000000000000000000000000000000000000000000000000000000" +
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        ↪  "0000000000000000000000000000000000000000000000000000000000000000" +
```

```
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000" +
        ↪ 00000000000000000000000000000000" +
        "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000" +
        ↪ 00000000000000000000000000000000" +
        "000000000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adeed713bae79d606" +
        ↪ 041406df26169c35df63cf972c94be1" +
        "000000000000000000000000000000011bc8afe71676e6730702a46ef817060249cd06cd82e6981085012ff6d013aa44" +
        ↪ 70ba3a2c71e13ef653e1e223d1ccfe9" +
        "000000000000000000000000000000013a3de1d25380c44ca06321151e89ca22210926c1cd4e3c1a9c3aa6c709ab5fdd" +
        ↪ 00f8df19243ce058bc753ccf03424ed" +
        "000000000000000000000000000000001657dbebf712cbda6f15d1d387c87b3fb9b386d5d754135049728a2a856ba2944" +
        ↪ c741024131a93c78655fdb7bfe3c803" +
        "000000000000000000000000000000000137232f722e38e084611ba67d2e28a3b8c73c13f20b6bb4c22141115bd43cdeb5" +
        ↪ 55861335f2a75d7cb418eb505341a2f" +
        "000000000000000000000000000000000153bdd82d3d9b76d1cab9d087654652ab1451f5fef4f449273d81211d88891fc5" +
        ↪ 3f131f98e2c3b4cb8c937b7d3a39bf2" +
        "000000000000000000000000000000000008d8c4a16fb9d8800cce987c0eadbb6b3b005c213d44ecb5adeed713bae79d606" +
        ↪ 041406df26169c35df63cf972c94be1" +
        "000000000000000000000000000000000084486ebc81878331aab7d6f53ca3c773fda7b181b56a93e5ee0bfa189afbb7fd" +
        ↪ 7a05c5bea35ec1054c0e1ddc2e2dac2" +
        "000000000000000000000000000000000013a3de1d25380c44ca06321151e89ca22210926c1cd4e3c1a9c3aa6c709ab5fdd" +
        ↪ 00f8df19243ce058bc753ccf03424ed" +
        "000000000000000000000000000000001657dbebf712cbda6f15d1d387c87b3fb9b386d5d754135049728a2a856ba2944" +
        ↪ c741024131a93c78655fdb7bfe3c803" +
        "000000000000000000000000000000000137232f722e38e084611ba67d2e28a3b8c73c13f20b6bb4c22141115bd43cdeb5" +
        ↪ 55861335f2a75d7cb418eb505341a2f" +
        "000000000000000000000000000000000153bdd82d3d9b76d1cab9d087654652ab1451f5fef4f449273d81211d88891fc5" +
        ↪ 3f131f98e2c3b4cb8c937b7d3a39bf2"
    )
test_pairing(data1)
test_pairing(data2)
test_pairing(data3)
```

Recommendation: EIP Specification Update. The specification should be updated to explicitly define:

- How infinity points affect multi-pairing products.
- Whether infinity points should be rejected at the precompile level.
- Clear mathematical properties that must be preserved.
- Comprehensive test vectors covering infinity point cases.

This clarification ensures consistent behavior across all Ethereum clients, preventing potential network splits and validator slashing risks.

3.2 Informational

3.2.1 Withdrawals: Recommend to change the example code in EIP7002 to prevent fee slippage

Submitted by Haxatron, also found by Haxatron

Severity: Informational

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

Description: In EIP-7002#fee-overpayment the recommended code to check for the fee is as follows:

```
function addWithdrawal(bytes memory pubkey, uint64 amount) private {
    assert(pubkey.length == 48);

    // Read current fee from the contract.
    (bool readOK, bytes memory feeData) = WithdrawalsContract.staticcall('');
    if (!readOK) {
        revert('reading fee failed');
    }
    uint256 fee = uint256(bytes32(feeData));

    // Add the request.
    bytes memory callData = abi.encodePacked(pubkey, amount);
    (bool writeOK,) = WithdrawalsContract.call{value: fee}(callData);
    if (!writeOK) {
        revert('adding request failed');
    }
}
```

However, as noted in the "User Level Considerations" section of Dedaub audit - EIP7002.pdf, the fee change can be drastic. It introduces developers who blindly follow the spec to slippage risks.

Recommendation: It is recommended to modify the example code in EIP-7002#fee-overpayment, to prevent slippage:

```
function addWithdrawal(bytes memory pubkey, uint64 amount, uint256 maxFee) private {
    assert(pubkey.length == 48);

    // Read current fee from the contract.
    (bool readOK, bytes memory feeData) = WithdrawalsContract.staticcall('');
    if (!readOK) {
        revert('reading fee failed');
    }
    uint256 fee = uint256(bytes32(feeData));
    require(fee <= maxFee, "Maximum fee exceeded!")
    // Add the request.
    bytes memory callData = abi.encodePacked(pubkey, amount);
    (bool writeOK,) = WithdrawalsContract.call{value: fee}(callData);
    if (!writeOK) {
        revert('adding request failed');
    }
}
```

3.2.2 Weak subjectivity calculation was not updated for Electra

Submitted by franfran

Severity: Informational

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

Summary: Since EIP-7251, the validator effective balance is allowed to go over 32 ETH. This changes the calculation of the churn limit which is now balance-based, as well as the weak subjectivity period calculation which directly depends on it.

Finding Description: EIP-7251 modifies the `get_validator_churn_limit` function since all validator won't have the same weight once this EIP becomes active. It also says to use this modified function in the `compute_weak_subjectivity_period` function in order to update it. This is explained in the 20 points of the consensus layer specification. More particularly, the point 5. explains to "*Modify `get_validator_churn_limit` to depend on the validator weight rather than the validator count.", and later the 13. which says "Modify `compute_weak_subjectivity_period` to use the new churn limit function*".

The wrong calculation of the validator churn limit doesn't affect the deposit or withdrawal queue position of validators, and it is believed that no consensus-related operation is touched. Thus, the impact is low.

The potentially highest impact is for Teku, which shutdown itself if the weak subjectivity is not within the finalized checkpoint. Fortunately, most checkpoints are final within 2 epochs, and a weak subjectivity period of 1 epoch is just **extremely unlikely** given the current Ethereum validators count (+1.000.000!). For an idea of the expected WS period, check the values that have been generated from the script of the consensus-specs.

Impact Explanation: Users will have a wrong idea of what checkpoints are safe to use, since the weak subjectivity calculation is currently too pessimistic.

Likelihood Explanation: All of the consensus clients didn't implemented the changes because it was missing in the consensus-specs.

Recommendation: Change the way the churn limit is calculated by replacing the validator count to the total validator balance, and maybe normalize the numerator by dividing it by the `MIN_ACTIVATION_BALANCE`?

3.2.3 EIP-7702: Clarify the restriction on `y_parity`

Submitted by Haxatron

Severity: Informational

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

Description: In the current EIP-7702 spec, the instruction to process a EIP-7702 transaction is noted as follows:

EIP-7702:

> 1. Verify the chain id is either 0 or the chain's current ID.
> 2. Verify the nonce is less than `2**64 - 1`.
> 3. `authority = ecrecover(keccak(MAGIC || rlp([chain_id, address, nonce])), y_parity, r, s)`
>   - `s` value must be less than or equal to `secp256k1n/2`, as specified in EIP-2.
> 4. Add authority to accessed_addresses (as defined in EIP-2929.)
> 5. Verify the code of authority is either empty or already delegated.
> 6. Verify the nonce of authority is equal to nonce. In case authority does not exist in the trie, verify that nonce is equal to 0.
> 7. Add `PER_EMPTY_ACCOUNT_COST - PER_AUTH_BASE_COST` gas to the global refund counter if authority exists in the trie.
> 8. Set the code of authority to be `0xef0100 || address`. This is a delegation designation.
> 9. As a special case, if address is `0x0000000000000000000000000000000000000000` do not write the designation. Clear the account's code and reset the account's code hash to the empty hash `0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470`.
> 10. Increase the nonce of authority by one.

However in Step 3, it is missing the restrictions in `y_parity` implemented in all execution clients. `y_parity` is checked to be only 0 and 1. It is recommended to clarify this restriction as a new execution client implementing the spec might implement EIP-7702 using the spec.

Recommendation: In Step 3, clarify the restriction of `y_parity`:

> 3. `authority = ecrecover(keccak(MAGIC || rlp([chain_id, address, nonce])), y_parity, r, s)`
>   - `s` value must be less than or equal to `secp256k1n/2`, as specified in EIP-2.
>   - `y_parity` value must be equal to 0 or 1