



MatterLabs – Verifier

Smart Contract Security
Assessment

Prepared by: Halborn

Date of Engagement: July 12th, 2023 – July 20th, 2023

Visit: Halborn.com

DOCUMENT REVISION HISTORY	2
CONTACTS	2
1 EXECUTIVE OVERVIEW	3
1.1 INTRODUCTION	4
1.2 ASSESSMENT SUMMARY	4
1.3 SCOPE	5
1.4 TEST APPROACH & METHODOLOGY	6
2 RISK METHODOLOGY	7
2.1 EXPLOITABILITY	8
2.2 IMPACT	9
2.3 SEVERITY COEFFICIENT	11
3 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	13
4 MANUAL TESTING	13
5 AUTOMATED TESTING	24
5.1 STATIC ANALYSIS REPORT	25
Description	25
Slither results	25
5.2 AUTOMATED SECURITY SCAN	27
Description	27
MythX results	27

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	07/19/2023
0.2	Document Updates	07/20/2023
0.3	Draft Review	07/21/2023
1.0	Remediation Plan	09/15/2023
1.1	Remediation Plan Review	09/15/2023
1.2	Remediation Plan Update	10/04/2023
1.3	Remediation Plan Update Review	10/05/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com



EXECUTIVE OVERVIEW



1.1 INTRODUCTION

This assessment was entirely focused on the new version of zkSync verifier which is a modified version of the Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge (PLONK) to optimize the proof system for zkSync Era circuits.

MatterLabs engaged Halborn to conduct a security assessment on their verifier smart contract beginning on July 12th, 2023 and ending on July 20th, 2023. The security assessment was scoped to the smart contract provided to the Halborn team.

Moreover, MatterLabs engaged Halborn to review a small update on their verifier smart contract (making the aggregative part to be optional depending on the verification keys) beginning on October 2nd, 2023 and ending on October 4th, 2023.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues within the smart contract

In summary, Halborn did not identify any security risks within the verifier smart contract.

1.3 SCOPE

1. IN-SCOPE:

The security assessment was scoped to the following [smart contract](#):

- [ethereum/contracts/verifier/Verifier.sol](#)

Commit ID: [f783f571e16a1b1adddb13db45db741f83b94812](#)

And also to the following commit ID including the new update:

Commit ID: [4416321060c3ffac7d1d279602ca81809a259c18](#)

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. ([solgraph](#))
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. ([MythX](#))
- Static Analysis of security for scoped contract, and imported functions. ([Slither](#))
- Testnet deployment. ([Foundry](#))

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A)	1
	Specific (AO:S)	0.2
Attack Cost (AC)	Low (AC:L)	1
	Medium (AC:M)	0.67
	High (AC:H)	0.33
Attack Complexity (AX)	Low (AX:L)	1
	Medium (AX:M)	0.67
	High (AX:H)	0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = \max(m_I) + \frac{\sum m_I - \max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = \min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	0



MANUAL TESTING



The main goal of the manual testing performed during this assessment was to test that the verifier is properly working to verify the zk proofs generated by the zkSync Era circuits, focusing on the following points/scenarios:

Test	Result
Check that using any valid proof, the verifier is able to properly verify it and returns a true as a result	Pass

[illegible]

MANUAL TESTING

Test	Result
Check that verifier reverts with proof is invalid if a maliciously forged serialized proof is sent	Pass

[illegible]

Test	Result
Check that verifier reverts with proof is invalid if less than 44 words for serialized proof is sent	Pass

[illegible]

Test	Result
Check that verifier reverts with proof is invalid if less than 4 words for recursive aggregation input is sent	Pass

[illegible]

Test	Result
Check that verifier returns a true for a public input with dirty bits over Fr mask	Pass

[illegible]

Test	Result
Check that the verifier returns a true having elliptic curve points over modulo	Pass

```
[
  (6980) PRECOMPUTE::ecmd(1331481874871893334779197469607874446826148727324855612183283589578126862798, 13494143176161086751279221986335628979455862868313586262779787732264449104635, 544388980813155121095
[staticcall]
  [
    14176619311897551523779426617675541851051754696965561673323857280623868374991, 942805240748525452694533683433685256822149357167157791521676738985137167172
    (150) PRECOMPUTE::ecmd(14176619311897551523779426617675541851051754696965561673323857280623868374991, 942805240748525452694533683433685256822149357167157791521676738985137167172, 4254236497062591530685
11994090[15685149831259581520659756488638979142212238017408304576848229546] [staticcall]
    81922286259707277820487314439987835740363113659220745992292767384820944, 1405733899972745658578062826929456930226029625678959458586547715828029287259
    (6980) PRECOMPUTE::ecmd(5558280385751162852868387327352787364885779562751792429916918431248322115, 110499225148138972168491647526375723698288977736869249044998037679645888324, 162543798883580413159
[staticcall]
  [
    1937196779573814775269798768338473832196883881343915638425221941727574348135, 1089649388375487236466545451076798374840893677525290438227227386967811734964
    (150) PRECOMPUTE::ecmd(1937196779573814775269798768338473832196883881343915638425221941727574348135, 1089649388375487236466545451076798374840893677525290438227227386967811734964, 816222286250976727378
1457338399774565897862632945698622120226256798945685067715828029287259) [staticcall]
    19783988117207258939244146892242154715214074715817024729455598741524, 412941696707171988892968611665158313761385366225481923639672054919639522
    (6980) PRECOMPUTE::ecmd(10856868626458774927132263160471517755978915825982835215838391666972843, 858935648709842441004659165891219488880131785254312624515928607803222345483, 2859440854482125460292
[staticcall]
  [
    435112613765742482584633994784536646781086087877741174697924489665768775869, 2149174878182179748286441203356136218320634541729553962942759848694863617
    (150) PRECOMPUTE::ecmd(435112613765742482584633994784536646781086087877741174697924489665768775869, 2149174878182179748286441203356136218320634541729553962942759848694863617, 178789808117267258035
12941669707171088892968611665158313761385366225481923639672054919639522) [staticcall]
    5210885641982981774897173835878080420826652264374347178445317795770194, 988803691926294293755104686276382217805637287880513695664431237740486939
    (6980) PRECOMPUTE::ecmd(226110836220837305555151428620446035983283181113150181569227811040171929804, 16576514235314335585292407242414297389775852406317563312380204893738838912, 791499858868497375234
[staticcall]
  [
    28907480413867592745988980406794669510684348060831069948228634228568432604679, 901823855788213367405135187734467489611554681074667305571347762925130749570
    (150) PRECOMPUTE::ecmd(28907480413867592745988980406794669510684348060831069948228634228568432604679, 901823855788213367405135187734467489611554681074667305571347762925130749570, 36230885641982988179409
886369195264293976104662768021780562722780651369566442327740486939) [staticcall]
    118293584061105747426254212328059782838476836397836483179891719937514227276, 175520379880745439163279773852558031436452583174449947783393969140693580439297
    (150) PRECOMPUTE::ecmd(118293584061105747426254212328059782838476836397836483179891719937514227276, 175520379880745439163279773852558031436452583174449947783393969140693580439297, 178528379880745439163279773852558031436452583174449947783393969140693580439297) [staticcall]
    114954395580511821795580481980874781293179238931632284479816387307474941466835, 181405894659459087268956824865255273155907906864975641879289804855104376660
    (6980) PRECOMPUTE::ecmd(114954395580511821795580481980874781293179238931632284479816387307474941466835, 181405894659459087268956824865255273155907906864975641879289804855104376660, 46897395140675352085
468973951406753520852640769868507896381947521129204945477798459781705, 48116381302085265737328214199452550470452256395791326487965858072808081065
    (150) PRECOMPUTE::ecmd(114954395580511821795580481980874781293179238931632284479816387307474941466835, 181405894659459087268956824865255273155907906864975641879289804855104376660, 46897395140675352085
3778841509834486853847693227125382125658980928232688157786571526260718) [staticcall]
    47747746850845594613857740417398587864290529535618408725415008051232, 23945842289320218212015430816619555984116926553774137707692754087339451
    (6980) PRECOMPUTE::ecmd(16954586249736859288323125028096150913945599547384354941191381394834068, 77186637167944638637038235197981784115075292342767436298662633617841923857, 4106880893991154557899
[staticcall]
  [
    201449646167923831807141083599942059713475374808519749776780798085487267749, 1274724232758921369572062570798998657258449411334678080412222563622414945252
    (150) PRECOMPUTE::ecmd(201449646167923831807141083599942059713475374808519749776780798085487267749, 1274724232758921369572062570798998657258449411334678080412222563622414945252, 477117740658845594613
2394584022983201212701643808166185558984116952563774137707692754087339451) [staticcall]
    6366892224664661340787423957330746522784627697513708020760720817484669, 151234871843646249355682968224985420586559218145834761807207706767459302904
    (6980) PRECOMPUTE::ecmd(631484125789832639474993416884710586271206903850856776749426071503220429, 127262807007041743753592094979981820947420838770155024094245819658103406226, 1905845588668796610
[staticcall]
  [
    166381862037955940583384290675044200785415707646191904132945746435191188875, 7564118061772911898487189237498541107994661559555711543172308048398398
    (150) PRECOMPUTE::ecmd(166381862037955940583384290675044200785415707646191904132945746435191188875, 7564118061772911898487189237498541107994661559555711543172308048398398, 51048572224636651340878
234871475636242936562768212698425265855818165534745109207908647459302904) [staticcall]
    76920837083980591432395839494286599473314493272901714213789723833938, 1722185474659232938045329495246690446683927756360558420423995785041085929830
    (6980) PRECOMPUTE::ecmd(631484125789832639474993416884710586271206903850856776749426071503220429, 127262807007041743753592094979981820947420838770155024094245819658103406226, 184754521677106392937
[staticcall]
  [
    1121293802550457013487585789382643513513801783944807519404552349632414396, 965404646526740451620710406823695051721615260757079281816395351116405359
    (150) PRECOMPUTE::ecmd(1121293802550457013487585789382643513513801783944807519404552349632414396, 965404646526740451620710406823695051721615260757079281816395351116405359, 718663721694453057682851197817841159752923476754369866233617841923807
80363242347158731292159483598297076481368836382109472857492959764455954, 184668121882803526127855974091888478984173823602106887856078527304558504
    (113080) PRECOMPUTE::ecmd(119778318577411999666337808041668186655446805235299259599686316212087375, 191817156047779586831691356814532739525866230921154499188151555994408170807, 805634, 208578469992308571594457076222829481370756395785188669051999238565852781, 40823678758634368133220403145435568316851327693401208105741075214128093531, 849565839231234321417684973247489272438418190887263
    [
      true
    ]
    (0) console::log(true) [staticcall]
  ]
}
```

MANUAL TESTING

Test	Result
Check that the verifier returns a true, having Fr over modulo	Pass

[illegible]

Test	Result
Check that verifier reverts with proof is invalid if more than 1 public inputs is sent	Pass

```
(FAIL: Reason: loadProof: Proof is invalid) test_3x() (gas: 152154)
Traces:
[4648928] PoC::iseth()
├── [0] Whi::add: (170) [staticcall]
│   ├── 0x05FAFED02C7814018F364224C75A036e40822F52
│   └── [2237390] new Verifier0xC671865D4017F316EC6d6F64422e1e82c47c246
│       └── 11175 bytes of code
└── = ()

[152154] PoC::test_3x()
├── [8192] verify::verify() ([123865576579298458661564742432597860723927871731166703813954773821444496835, 123865576579298458661564742432597860723927871731166703813954773821444496255], [19677261140296324268841822849438449728033663396536485256077174570736611189, 1051298659319169461329284667451118893936788614958156911589972761418808695815934, 2475890346987583188421134436487231235104044955782344185287251246249, 169589289928624433851247292765817481444557656274760731722
984380526082121717722083116095851969, 507691167610898694908587819429756918189438087585636863238886197258855137, 133148187487189334779197469087044466826187272405651218328589587812862790, 134941431761618887
35, 32047266427348318686745276309797671681188882893538442406128643589990, 11121606122324365974314598882773834353118841086452707403415643711389183, 55682083065176138528586897238275734885779627617
73794939897724869244988767474588824, 18846489245297452931322616817157657939162599282823152839414448972849, 880635487809244118046515589319488981173826316224651572869780322264543, 47934141854
4997808, 211955653112275823428399841181964351388886397671235182647298261639271767, 103673151477768693194359676476507860222852485982378589595341864364463873955, 752236438958584916374582552461934261484240942
49327280150595866471551168493272457412514739738086, 2397780477957865788971385114207455058411540786842659374513838846804, 798136984259851269753382429505489623788211221967129716391285108595228, 13057
932247912558459, 1451926545284782780145234763798839818448376421185648702345853234516758, 6446835858515541265434498241943145152804376218714477218798838311544924142, 60524047220488116572765298716088111057
58416429416663938780479597212034751816786422358152445664, 157996377493692583258925892321678314874816585861383767856108138961910924, 11182289428253691645481086763476240649575732195866388044465233895897905
242479418974852209785, 14226476522499712819025047214062436185899524698649522824365471225814295, 114209415583123928213589257166484711648411807187785976880796238848461828, 10779437152338586118852174582
93394181149727827118189180248426497245784454145462240147328972, 4128509312972916367824667354617789793747358000904782747769401528224, 178404403857344942737435276007588839647718881077874434383806685
36472728848837209695737629895938, 16131887266476658452687283806821832624565337813857764895959919081481611, 1168259875164599647080114099355519853624461862884236187262718720579614226137, 2198838121551292256861
6591911811805957737812978578612841962512223707633145148443484495981, 169548862437638928837458971394597478783847411013513941834068, 77186617234740453805780382351197817841587527247647632986
24732983858565277454426071582320429, 1272628078078414743763592804877993182947432883781558240424581945818340620), [2257208268264499594444385474309939742774212892275774525482327892625584, 989121870191474
2538, 16188804989048318109493583376791797672882599586082624530949286765821563, 326109356769626567598585311767239987238003146738631772984817549983765385], [staticcall]
└── "loadProof: Proof is invalid"
    └── "loadProof: Proof is invalid"
```

Test	Result
Check that verifier reverts with proof is invalid if empty public inputs is sent	Pass

```
(FAIL: Reason: loadProof: Proof is invalid) test_6x() (gas: 122538)
Traces:
[4648928] PoC::iseth()
├── [0] Whi::add: (170) [staticcall]
│   ├── 0x05FAFED02C7814018F364224C75A036e40822F52
│   └── [2237390] new Verifier0xC671865D4017F316EC6d6F64422e1e82c47c246
│       └── 11175 bytes of code
└── = ()

[122538] PoC::test_6x()
├── [8192] verify::verify() ([19677261140296324268841822849438449728033663396536485256077174570736611189, 1051298659319169461329284667451118893936788614958156911589972761418808695815934, 24758903469875831884
1695892899286244338512472927658174814445576562747607317222983936788614958156911589972761418808695815934, 2475890346987583188421134436487231235104044955782344185287251246249, 169589289928624433851247292765817481444557656274760731722
984380526082121717722083116095851969, 507691167610898694908587812862790, 134941431761618887127922198633562897945586204031358626277787727264641840485, 32047266478405346687452786398792676186111888028059383442406128643589990, 111216061232
13919189, 5558308306517613852858689723735278736488577956276170242971691841240322115, 110499225163183897216849417526375726958280777268082498847988278794588824, 18846489245297452931322616817157657939162599282823152839414448972849, 880635487809244118046515589319488981173826316224651572869780322264543, 47934141854
5897808, 211955653112275823428399841181964351388886397671235182647298261639271767, 103673151477768693194359676476507860222852485982378589595341864364463873955, 752236438958584916374582552461934261484240942672122319635416857769801887893, 2192624073662318204889449322728015059586644715511684943247564912514739738086, 2397780477957865788971385114207455058411540786842659374513838846804, 798136984259851269753382429505489623788211221967129716391285108595228, 13057
932247912558459, 1451926545284782780145234763798839818448376421185648702345853234516758, 6446835858515541265434498241943145152804376218714477218798838311544924142, 60524047220488116572765298716088111057
58416429416663938780479597212034751816786422358152445664, 157996377493692583258925892321678314874816585861383767856108138961910924, 1118228942825369164548108676347624064957573219586638804446523389589790525, 57648984114833885614744866781898417487598789211192487418994892098785, 1422647652224969312019628774148992436165858957428456214287
718728507880796238848461828, 1077943715223435811856216310636173892394247797470762632817742468347341, 3454254797319241181149772621151093186264254972457364843145624047422872, 4128509312972916367824667354617789793747358000904782747769401528224, 178404403857344942737435276007588839647718881077874434383806685
36472728848837209695737629895938, 16131887266476658452687283806821832624565337813857764895959919081481611, 1168259875164599647080114099355519853624461862884236187262718720579614226137, 219883812155129225686151748512392343661299936498658480752626196392458065, 1265391918110206957737012989572051328411026521222707633145148443484495981, 169548862437638928837458971394597478783847411013513941834068, 77186617234740453805780382351197817841587527247647632986
24732983858565277454426071582320429, 1272628078078414743763592804877993182947432883781558240424581945818340620), [2257208268264499594444385474309939742774212892275774525482327892625584, 9891218701914742538, 16188804989048318109493583376791797672882599586082624530949286765821563, 326109356769626567598585311767239987238003146738631772984817549983765385], [staticcall]
└── "loadProof: Proof is invalid"
    └── "loadProof: Proof is invalid"
```

Test	Result
Check that verifier reverts with proof is invalid if more than 44 words for serialized proof is sent	Pass

```
(FAIL: Reason: loadProof: Proof is invalid) test_4x() (gas: 152222)
Traces:
[4648928] PoC::iseth()
├── [0] Whi::add: (170) [staticcall]
│   ├── 0x05FAFED02C7814018F364224C75A036e40822F52
│   └── [2237390] new Verifier0xC671865D4017F316EC6d6F64422e1e82c47c246
│       └── 11175 bytes of code
└── = ()

[152222] PoC::test_4x()
├── [8192] verify::verify() ([123865576579298458661564742432597860723927871731166703813954773821444496835, 123865576579298458661564742432597860723927871731166703813954773821444496255], [19677261140296324268841822849438449728033663396536485256077174570736611189, 1051298659319169461329284667451118893936788614958156911589972761418808695815934, 2475890346987583188421134436487231235104044955782344185287251246249, 169589289928624433851247292765817481444557656274760731722
984380526082121717722083116095851969, 507691167610898694908587812862790, 134941431761618887127922198633562897945586204031358626277787727264641840485, 32047266478405346687452786398792676186111888028059383442406128643589990, 1112160612324365974314598882773834353118841086452707403415643711389183, 5558208306517613852858689723735278736488577956276170242971691841240322115, 110499225163183897216849417526375726958280777268082498847988278794588824, 18846489245297452931322616817157657939162599282823152839414448972849, 880635487809244118046515589319488981173826316224651572869780322264543, 47934141854
5897808, 211955653112275823428399841181964351388886397671235182647298261639271767, 103673151477768693194359676476507860222852485982378589595341864364463873955, 752236438958584916374582552461934261484240942672122319635416857769801887893, 2192624073662318204889449322728015059586644715511684943247564912514739738086, 2397780477957865788971385114207455058411540786842659374513838846804, 798136984259851269753382429505489623788211221967129716391285108595228, 13057
932247912558459, 1451926545284782780145234763798839818448376421185648702345853234516758, 6446835858515541265434498241943145152804376218714477218798838311544924142, 60524047220488116572765298716088111057
58416429416663938780479597212034751816786422358152445664, 157996377493692583258925892321678314874816585861383767856108138961910924, 1118228942825369164548108676347624064957573219586638804446523389589790525, 57648984114833885614744866781898417487598789211192487418994892098785, 1422647652224969312019628774148992436165858957428456214287
718728507880796238848461828, 1077943715223435811856216310636173892394247797470762632817742468347341, 3454254797319241181149772621151093186264254972457364843145624047422872, 4128509312972916367824667354617789793747358000904782747769401528224, 178404403857344942737435276007588839647718881077874434383806685
36472728848837209695737629895938, 16131887266476658452687283806821832624565337813857764895959919081481611, 1168259875164599647080114099355519853624461862884236187262718720579614226137, 219883812155129225686151748512392343661299936498658480752626196392458065, 1265391918110206957737012989572051328411026521222707633145148443484495981, 169548862437638928837458971394597478783847411013513941834068, 77186617234740453805780382351197817841587527247647632986
24732983858565277454426071582320429, 1272628078078414743763592804877993182947432883781558240424581945818340620), [2257208268264499594444385474309939742774212892275774525482327892625584, 9891218701914742538, 16188804989048318109493583376791797672882599586082624530949286765821563, 326109356769626567598585311767239987238003146738631772984817549983765385], [staticcall]
└── "loadProof: Proof is invalid"
    └── "loadProof: Proof is invalid"
```


MANUAL TESTING

Test	Result
Check that verifier reverts with proof is invalid if empty serialized proof is sent	Pass

Test	Result
Check that verifier reverts with proof is invalid if more than 4 words for recursive aggregation input is sent	Pass

[illegible]

Test	Result
Check that verifier reverts with proof is invalid if empty recursive aggregation input is sent	Pass

[illegible]

Test	Result
Check that verifier reverts with proof is invalid if elliptic curve point at infinity is sent within the serialized proof	Pass

```
[FAIL: Reason: loadProof: Proof is invalid] test_2v2() (gas: 133267)
Traces:
[444958] PcU::setUp()
  ↳ [0] VM::code(170) [staticcall]
    ↳ 0x0b9AF072C781481BF364224C76A0364A0832F52
    ↳ [2237390] = new Verifier0x0bC67186504017F316C666F4422e1e82c47c246
      ↳ = 11178 bytes of code
      ↳ = ()

[123667] PcU::test_2v2()
  ↳ [1819] Verifier::verify((112386557657929845866156472423259786723927871731166783813954973021444496256), [0, 0, 24788893469873883188443211365436487397212355184643697993434185539785124249, 10958865897206244338
    1, 884884248573286694156549552128478897418788862668210717227938114984581969, 58769116741889849474855781942975498181894386745876538863228886919255855157, 12314818748738334779154980767446682614877373485561
    2897945628681386626779787732264494184835, 32842766427840834866874527863978726178611888028059383424861286635899980, 1111681222924245974217459888273783439513411884186548276740315434711389183, 55532803305
    4822115, 1184972518138971476237572968288077368087240449880787964588324, 1088368680245308745922122610484717657889158269928362158380912668072849, 88085824076904341188425910589319488880175826
    98944088363399437898584745157455818849287028, 2113850531224725524283998411819453138886250212231559247209261639271767, 183473164477768923194289774726846722282438592378589534188426446397895, 722
    857769841809839, 21296248736623818284889449327208156559866447155119684932475491425146738086, 2398778647796786578897138511420745565841154078684626897451938138388646804, 79813698425995142692763232424595458962
    9888861223145611405381708989138876785186224791265066238, 145816294852847827289149384753708889818144387662218864870238488383234716736, 64688833985155618657464904194314518938481702187140771079888831154493416
    6274943264454447238, 17878939239465385188416642914664389380749592128478131878423281244564, 15794677389942332588920622474783187434848586313874785618813891518934, 111822863235894548188476178
    4833858561746486671889847148789785972113928479189948092897878, 1422647652294691281962504721406243619859595244686049522823567412296814295, 1142096415538123912681238925716645471164841180718775867685972389
    247974787621387974246847341, 3454245479319341101149772821151891818626482546724578454345602240417238972, 4128890112973291636792476697838681770797974473888808678274776696812872241, 178648448387436492797436
    4748124192737874443226549362765794971344772384837209409728698938, 1613138724647445845456728385623182454537813877634049593980148181165, 11682987516549947808114809285543893624418628542618732072
    661299364858684625623617693949345685, 12653919181102059577378129897285128419826521122237076314531684436484594981, 1695498862437638892083221831250280961589139455954737842549411013513941834868, 77186637169445
    67, 6314861167898126591774991418847185862712869838888677674742468715882282429, 127262878078417477638720497798182094742883877818582489424819681834862261, 12257782682624849749944438847438979747274212892272
    144639276174429774159411959331328491756259, 161888449789484381894939833787157160723825978469822624538949286705821563, 3281893557749626557989551176725988732888314747386377294847549837453851] [static
    call]
  ↳ "loadProof: Proof is invalid"
  ↳ "loadProof: Proof is invalid"
```

Test	Result
Check that the verifier reverts with invalid quotient evaluation if an invalid public input is used	Pass

```
[FAIL: Reason: invalid quotient evaluation] test_5x() (gas: 149561)
Traces:
[444958] PcU::setUp()
  ↳ [0] VM::code(170) [staticcall]
    ↳ 0x0b9AF072C781481BF364224C76A0364A0832F52
    ↳ [2237390] = new Verifier0x0bC67186504017F316C666F4422e1e82c47c246
      ↳ = 11178 bytes of code
      ↳ = ()

[149561] PcU::test_5x()
  ↳ [2758] Verifier::verify((112386557657929845866156472423259786723927871731166783813954973021444496256), [196772611402962426884138238494384697280839643396536485268771764570736611109, 10511986593191694613920
    478893449873583188443211365425687372123518684695729624638581247239756877481444957566224736877317220999881632487461, 8848842485248523286694166695522128478897410738626682107172
    1894389748378368632358691925585517, 1231481874873833477915498076744668261487737348556, 32842766427840834866874527863978726178611888028059383424861286635899980, 1111681222924245974217459888273783439513411884186548276740315434711389183, 55532803305
    4822115, 1184972518138971476237572968288077368087240449880787964588324, 1088368680245308745922122610484717657889158269928362158380912668072849, 8808582407690434118842591058931948880175826
    98944088363399437898584745157455818849287028, 2113850531224725524283998411819453138886250212231559247209261639271767, 183473164477768923194289774726846722282438592378589534188426446397895, 722
    857769841809839, 21296248736623818284889449327208156559866447155119684932475491425146738086, 2398778647796786578897138511420745565841154078684626897451938138388646804, 79813698425995142692763232424595458962
    9888861223145611405381708989138876785186224791265066238, 145816294852847827289149384753708889818144387662218864870238488383234716736, 64688833985155618657464904194314518938481702187140771079888831154493416
    6274943264454447238, 17878939239465385188416642914664389380749592128478131878423281244564, 15794677389942332588920622474783187434848586313874785618813891518934, 111822863235894548188476178
    4833858561746486671889847148789785972113928479189948092897878, 1422647652294691281962504721406243619859595244686049522823567412296814295, 1142096415538123912681238925716645471164841180718775867685972389
    247974787621387974246847341, 3454245479319341101149772821151891818626482546724578454345602240417238972, 4128890112973291636792476697838681770797974473888808678274776696812872241, 178648448387436492797436
    4748124192737874443226549362765794971344772384837209409728698938, 1613138724647445845456728385623182454537813877634049593980148181165, 11682987516549947808114809285543893624418628542618732072
    661299364858684625623617693949345685, 12653919181102059577378129897285128419826521122237076314531684436484594981, 1695498862437638892083221831250280961589139455954737842549411013513941834868, 77186637169445
    67, 6314861167898126591774991418847185862712869838888677674742468715882282429, 127262878078417477638720497798182094742883877818582489424819681834862261, 12257782682624849749944438847438979747274212892272
    144639276174429774159411959331328491756259, 161888449789484381894939833787157160723825978469822624538949286705821563, 3281893557749626557989551176725988732888314747386377294847549837453851] [static
    call]
  ↳ "invalid quotient evaluation"
  ↳ "invalid quotient evaluation"
```

Test	Result
Check that the verifier reverts with pairing failure if an invalid recursive aggregative input is used	Pass

[illegible]



AUTOMATED TESTING



Description:

Slither results:

```
INFO:Detectors:
Parameter Verifier.verify_asm_0_revertWithMessage().in_verify_asm_0_revertWithMessage (src/verifier/Verifier.sol#147) is not in mixedCase
Parameter Verifier.verify_asm_0_revertWithMessage().reason_verify_asm_0_revertWithMessage (src/verifier/Verifier.sol#147) is not in mixedCase
Parameter Verifier.verify_asm_0_modeExp().value_verify_asm_0_modeExp (src/verifier/Verifier.sol#148) is not in mixedCase
Parameter Verifier.verify_asm_0_modeExp().power_verify_asm_0_modeExp (src/verifier/Verifier.sol#148) is not in mixedCase
Parameter Verifier.verify_asm_0_pointMulInputDest().v_verify_asm_0_pointMulInputDest (src/verifier/Verifier.sol#175) is not in mixedCase
Parameter Verifier.verify_asm_0_pointMulInputDest().dest_verify_asm_0_pointMulInputDest (src/verifier/Verifier.sol#175) is not in mixedCase
Parameter Verifier.verify_asm_0_pointMulInputDest().dest_verify_asm_0_pointMulInputDest (src/verifier/Verifier.sol#175) is not in mixedCase
Parameter Verifier.verify_asm_0_pointAddInputDest().p2_verify_asm_0_pointAddInputDest (src/verifier/Verifier.sol#188) is not in mixedCase
Parameter Verifier.verify_asm_0_pointAddInputDest().p2_verify_asm_0_pointAddInputDest (src/verifier/Verifier.sol#188) is not in mixedCase
Parameter Verifier.verify_asm_0_pointSubAssign().p1_verify_asm_0_pointSubAssign (src/verifier/Verifier.sol#196) is not in mixedCase
Parameter Verifier.verify_asm_0_pointSubAssign().p1_verify_asm_0_pointSubAssign (src/verifier/Verifier.sol#196) is not in mixedCase
Parameter Verifier.verify_asm_0_pointSubAssign().p2_verify_asm_0_pointSubAssign (src/verifier/Verifier.sol#196) is not in mixedCase
Parameter Verifier.verify_asm_0_pointSubAssign().p2_verify_asm_0_pointSubAssign (src/verifier/Verifier.sol#196) is not in mixedCase
Parameter Verifier.verify_asm_0_pointAddAssign().p1_verify_asm_0_pointAddAssign (src/verifier/Verifier.sol#247) is not in mixedCase
Parameter Verifier.verify_asm_0_pointAddAssign().p1_verify_asm_0_pointAddAssign (src/verifier/Verifier.sol#247) is not in mixedCase
Parameter Verifier.verify_asm_0_pointAddAssign().p2_verify_asm_0_pointAddAssign (src/verifier/Verifier.sol#247) is not in mixedCase
Parameter Verifier.verify_asm_0_pointAddAssign().p2_verify_asm_0_pointAddAssign (src/verifier/Verifier.sol#247) is not in mixedCase
```


5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the verifier contract and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

Report for Verifier.sol
<https://dashboard.mythx.io/#/console/analyses/38dc0fe1-0e22-4008-a6a8-2c44e81bfcf>

Line	SWC Title	Severity	Short Description
3	(SWC-103) FloatingPragma	Low	A floating pragma is set.

- No major issues found by Mythx.



THANK YOU FOR CHOOSING

 **HALBORN**

