Polygon zkEVM Security Review

December 2022 Engagement

Auditors

Alex Beregszaszi, Lead Security Researcher

Andrei Maiboroda, Lead Security Researcher

Christian Reitwiessner, Lead Security Researcher

Leonardo Alt, Lead Security Researcher

Paweł Bylica, Lead Security Researcher

Thibaut Schaeffer, Lead Security Researcher

Lucas Vella, Security Researcher

Miguel Palhas, Associate Security Researcher

Report Version 1

March 27, 2023

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

Polygon zkEVM is a new zk-rollup that provides Ethereum Virtual Machine (EVM) equivalence (opcode-level compatibility) for a transparent user experience and existing Ethereum ecosystem and tooling compatibility.

It consists of a decentralized Ethereum Layer 2 scalability solution utilising cryptographic zero-knowledge technology to provide validation and fast finality of off-chain transaction computations.

This approach required the recreation of all EVM opcodes for transparent deployment and transactions with existing Ethereum smart contracts. For this purpose a new set of tools and technologies were created and engineered, and are contained in this organization.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of 0xPolygonHermez 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 10 days in total (a total of 12 person weeks), Polygon engaged with Spearbit to review the Polygon zkEVM protocol. In this period of time a total of 32 issues were found.

During this engagement we had a set of distinct goals focusing on a number of different components:

1. Inspect test coverage of Ethereum state tests and evm-benchmarks.
2. Manual and tooling aided review of a number of state machines (in PIL).
3. Manual review of zkASM implementation of the EVM instructions.

Our researchers formed two groups:

- one focused on state machines and PIL (for the entire duration),
- the other focused on instruction implementations and testing assessment (for half of the total duration).

A comprehensive explainer of the extent of review is explained in the section about 6 for coverage.

We recommend a follow up engagement to increase review coverage.

Summary

Project Name	Polygon zkEVM
Repository	See section 6
Type of Project	Zero Knowledge EVM, zk-rollup
Audit Timeline	Dec 5th - Dec 16th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	7	7	0
Incompatibility	1	0	1
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	22	18	3
Total	**32**	**27**	**4**

5 Findings

5.1 Critical Risk

5.1.1 Gas calculation frequently uses unchecked free variables

Severity: Critical Risk

Context: zkevm-rom:calldata-returndata-code.zkasm#L73, zkevm-rom:calldata-returndata-code.zkasm#L225, zkevm-rom:calldata-returndata-code.zkasm#L329, zkevm-rom:utils.zkasm#L395, etc.

Description: An example in **opCALLDATACOPY** where unconstrained free variables are used to calculate the word-size cost:

```
GAS - 3 => GAS  :JMPN(outOfGas)
GAS - ${3*((C+31)/32)} => GAS    :JMPN(outOfGas) ; Arith
```

In some places, like **opRETURNDATACOPY** these are mostly performed using the arithmetic state machine:

```
;${3*((C+31)/32)}
C+31 => A
;(C+31)/32
A               :MSTORE(arithA)
32              :MSTORE(arithB)
                :CALL(divARITH)
$ => A          :MLOAD(arithRes1)
; Mul operation with Arith
; 3*((C+31)/32)
3               :MSTORE(arithA)
A               :MSTORE(arithB)
                :CALL(mulARITH)
$ => A          :MLOAD(arithRes1)
GAS - A => GAS  :JMPN(outOfGas)
```

Under tag **0.5.2.0** a number of instances remain, including **opCALLDATCOPY**, **opCODECOPY**, **opEXTCODCOPY**, and the **saveMemGAS** utility affecting memory expansion.

Polygon-Hermez: All unsafe computations has been fixed in PR #199 and PR #221.

Spearbit: Acknowledged.

5.1.2 Memory alignment malleability

Severity: Critical Risk

Context: zkevm-proverjs:sm_mem_align.js#L14

Description: The constant polynomial **BYTE_C4096** does not wrap around after value 255 as it should, allowing **inV** to have values above a byte, which may carry over and cause an **MSTORE8** to write a 1 before the given byte in memory.

Background: Solidity code generation always tries to align memory offsets (such as allocation is done at word-boundaries) and many other projects writing inline assembly are encouraged to do so. However, some people optimising for gas do not that, and also it cannot be known for certain that no unaligned case exists in Solidity's code generator.

Now imagine if "fake value" insertion is possible in unaligned **MLOAD**s, that means a malicious block builder could make a user lose or gain value in for example an AMM transaction or other kinds of swaps. Such transactions could be sandwiches to extract value. Cases of unaligned reads could be easily mapped out by transaction tracing and "searchers" could build up a database of proof patterns to modify for automatic execution of such malicious trades.

Recommendation: Wrap around after value 255.

```
@@ -11,7 +11,7 @@ const CONST_F = {
     BYTE2B: (i) => (i & 0xFF), // [0..255]

     // 0 (x4096), 1 (x4096), ..., 255 (x4096), 0 (x4096), ...
-    BYTE_C4096: (i) => (i >> 12), // [0:4096..255:4096]
+    BYTE_C4096: (i) => (i >> 12) & 0xFF, // [0:4096..255:4096]

     //    0 - 1023  WR256 = 0 WR8 = 0  i & 0xC00 = 0x000
     // 1024 - 2047  WR256 = 0 WR8 = 0  i & 0xC00 = 0x400
```

Polygon-Hermez: Fixed in PR #116.

Spearbit: Acknowledged.

5.1.3 Incorrect `BYTE` implementation

Severity: Critical Risk

Context: zkevm-rom:comparison.zkasm#L345

Description: In **`opBYTE`**, the argument range check **`31 - B => D      :JMPN(opBYTE0)`** is incorrect because this only checks **`B0`**. If any of **`B1-B7`** are non-zero, the result of **`opBYTE`** must be **`0`** but may be other value in this implementation.

Counter example:

- **`A = 0xa0a1a2a3a4a5a6a7a8a9b0b1b2b3b4b5b6b7b8b9c0c1c2c3c4c5c6c7c8c9d0d1`**,
- **`B = 0x100000001`**, i.e. **`B0 = 1`**, **`B1 = 1`**,
- The result should be **`0`** but is **`0xa0`**.

This case or similar are not covered by the Ethereum State Tests, but various implementations cover it via unit tests (like **`evmone`**).

Recommendation: Use **`:SUB`**.

Polygon-Hermez: Fixed in PR #211 and specific tests added in PR #165.

Spearbit: Acknowledged.

5.1.4 Test failures in `VMTests/vmPerformance/loopExp` (potential issue in `EXP`)

Severity: Critical Risk

Context: zkevm-testvectors

Description: Some Ethereum tests from **`VMTests/vmPerformance/loopExp.json`** are failing with the following error:

```
Input: /0xPolygonHermez/zkevm-testvectors/tools/ethereum-tests/eth-inputs/GeneralStateTests/VMTests/loo
↪  pExp_1.json
Start executor JS...
Error
Error: Program terminated with registers A, D, E, SR, CTX, PC, MAXMEM, zkPC not set to zero
```

The list of failing tests:

```
loopExp_1
loopExp_2
loopExp_3
```

Tests can be found in our internal repository. They are modified versions of Ethereum tests with lowered transaction gas limit and number of loop iterations decreased to fit into the 30M gas limit.

It is hard to diagnose the exact reason since testing infrastructure does not save trace log file (see *Trace log is not saved in case of "register not set to zero" error*) in case of such error (unlike in the case of **`newStateRoot does not match`** error).

We think this may signal a bug in the implementation of **`EXP`**.

Recommendation: Ensure these test can be executed and verify that **`EXP`** behaves correctly.

Polygon-Hermez: Fixed in PR #211 and improved zk-counters check in PR #212

5.1.5 Incorrect `SAR` implementation

Severity: Critical Risk

Context: zkevm-rom:comparison.zkasm#L488

Description: The implementation of **`opSAR`** is incorrect. In case **`A`** is "negative" (sign bit is 1) the correction of the result from **`SHRarithBit`** is incorrectly done by negation (**`0 - A`**).

Counter example: **`sar(0x80...01, 1)`** gives **`0xc0..01`** instead of **`0xc0..00`**:

```
sign(A) == 1
abs(A) == 0x7f..ff
abs(A) >> 1 == 0x3f..ff
0 - (abs(A) >> 1) == 0xc0..01
```

These tests should uncover the bug, but they are disabled:

```
[  FAILED  ] stShift.shiftCombinations
[  FAILED  ] stShift.shiftSignedCombinations
```

Recommendation: The correct implementation of **`SAR`** in the case of **`A`** being negative would be **`NOT(SHR(NOT(A), D))`** (**`A`** is the number to shift, **`D`** is the amount of bits to shift by), where **`NOT`** is bitwise negation, not arithmetic negation as in the current implementation.

We could use **`XOR`** and the sign bit to conditionally negate. If **`E`** contains the sign bit, then this would be (using functional and assignment notation):

```
MASK = SUB(0, E)
RESULT = XOR(SHR(XOR(A, MASK), D), MASK)
```

Polygon-Hermez: Fixed in PR #211 and test added in PR #165.

Spearbit: Acknowledged.

5.1.6 Insufficient check about division remainder

Severity: Critical Risk

Context: zkevm-rom:utils.zkasm#L501

Description: The **`divARITH`** subroutine uses the arithmetic state machine to perform division. The equation used is **`A * B + C = D * 2**256 + E`**, all of those variables are 256-bit unsigned numbers, but the equation is an equation on integers.

The subroutine computes **`E / A`** and **`E % A`** at the same time (both **`E`** and **`A`** are inputs to the subroutine and are assigned like that to the variables of the state machine).

The subroutine performs two checks before it invokes the state machine. Those are:

- division by zero (**`A == 0`**), it directly returns **`(0, 0)`**

- **`E < A`**: In this case, it directly returns **`(0, E)`**

Then it invokes the state machine, setting **`D = 0`**, **`B = ${E/A}`** (free input), **`C = ${E%A}`** (free input).

Choosing these free inputs makes the state machine succeed.

The problem is that the remainder **`${E%A}`** is only a proper remainder if it is less than **`A`**. If we for example invoke the state machine with **`B = ${E/A - 1}`**, **`C = ${(E%A) + A}`**, this will also satisfy the equation (as long as **`E/A >= 1`** and we don't get an arithmetic overflow).

In line 501, a comment states that this check "remainder < divisor" is performed after invoking the state machine, but the code actually performs a different check: It invokes the **`LT`** state machine on **`C`** and **`E`** and thus compares **`C < E`** instead of **`C < A`**.

This means a malicious prover can forge invalid division and modulo operations.

Note that due to the check against **`E < A`** before invoking the state machine, we only invoke the state machine in the case that **`A <= E`** and thus the invalid check **`C < E`** is never stricter than the correct check **`C < A`**, which resulted in this not being discovered in tests.

Recommendation: Change the code after line 501 from

```
    C => A ; reminder
    E => B ; divisor
```

to

```
    A => B ; divisor
    C => A ; remainder
```

Polygon-Hermez: Fixed in PR #205.

Spearbit: Acknowledged.

5.1.7 Underconstrained carry value in the binary state machine

Severity: Critical Risk

Context: zkevm-proverjs:binary.pil

Description: The value of **`cIn`** can be set arbitrarily by the prover when **`RESET`** is **`1`**. As a result, the prover can prove that **`0 + 0 == 1`** by setting **`cIn`** to 1.

The issue can be exploited as follows, for example for the first operation in the binary state machine:

```
// sm_binary.js:337
+ pols.cIn[0] = 1n;
```

Recommendation: Change the PIL constraint over **`cIn`**:

```
// binary.pil:82
- cIn' * (1 - RESET) = cOut * ( 1 - RESET' )
+ cIn' = cOut * ( 1 - RESET' )
```

With this change **`(RESET == 1) => (cIn == 0)`**.

Polygon-Hermez: Fixed in PR #137.

Spearbit: Acknowledged.

5.2 Incompatibility

5.2.1 Jumping into push-data is possible (deviation from mainnet)

Severity: Incompatibility

Context: zkevm-rom:flow-control.zkasm#L58-L59

Description: It is only checked that destination is 0x5b (**`JUMPDEST`**) byte, but not whether 0x5b is part of preceding **`PUSH`** instruction. Ethereum jumpdest-analysis mandates that it should not be possible to jump into push-data, which is a strong separation of code and data during execution.

While the requirement of the **`JUMPDEST`** opcode was inspired by easier transformation of EVM code to machine code (by enabling the detection of basic execution blocks), later this feature has been expected by developers and tools alike to avoid (malicious) bugs being hidden in data portions of the code.

If the zkEVM will allow such code to be valid, analysing deployed code will need modified tools, and the security researcher community must be made aware of this difference.

Example code that should fail:

```
#  Bytes      Opcode
#  Location
#  --------   ----------
#  00-01      PUSH1 0x01
#  02-03      PUSH1 0x00
#  04         SSTORE
#  05-06      PUSH1 0x0A (End of PUSH data location)
#  07         JUMP
#  08-0A      PUSH2 0x5B5B
#  0B         JUMPDEST
```

Bytecode: **`0x6001600055600A56615B5B5B`**

Recommendation: Exceptionally abort if jump destination is a 0x5b byte inside push-data.

Polygon-Hermez: This is a known difference between EVM and zkEVM implementation. It requires code **`JUMPDEST`** analysis at runtime which implies a high cost in zkEVM and it has been decided to not perform this check.

Spearbit: Acknowledged, but strongly recommend against.

5.3 High Risk

5.3.1 Transaction processing doesn't reject malleable signatures

Severity: High Risk

Context: load-tx-rlp.zkasm, process-tx.zkasm, ecrecover.zkasm

Description: Signatures with *high S values* were restricted in EIP-2 (i.e. the Homestead upgrade):

> All transaction signatures whose s-value is greater than secp256k1n/2 are now considered invalid. The ECDSA recover precompiled contract remains unchanged and will keep accepting high s-values; this is useful e.g. if a contract recovers old Bitcoin signatures.

The transaction processing code in **`processTx`** passes down the raw signature details to **`ecrecover`**, which only performs validation of **`r in [1, secp256k1n - 1]`** and **`s in [1, secp256k1n - 1]`**. Since the **`ecrecover`** code is used for the precompile, this is expected there, however for transaction processing the valid range is **`s in [1, secp256k1n / 2]`**.

See also the Appendix F of the Yellow Paper.

Recommendation: Reject such transactions.

Polygon-Hermez: Fixed in PR #201.

Spearbit: Acknowledged.

5.4 Medium Risk

5.4.1 Ethereum Test Suite is designed to be run on multiple versions and not a single one

Severity: Medium Risk

Context: zkevm-testvectors

Description: The Ethereum Test Suite is designed to support multiple protocol versions (called **`Network`**), ran from the genesis (called **`Frontier`**) in increments until the last release.

Some tests are enabled a) on specific old versions (e.g. only **`Frontier`**) because nobody updated them. b) starting specific new versions (e.g. **`London`**) because they rely on some new EVM features to make testing easier, but the feature they actually are designed to test were enabled from earlier upgrades.

The zkEVM is tied to the **`Berlin`** release only and may miss out on some test cases. We have updated some test cases to be enabled on Berlin too, but a comprehensive review of the test suite is needed.

Recommendation: Ensure that all relevant tests are run on the Berlin version.

Polygon-Hermez: We will review the full Ethereum test suite to identify missing test regarding Berlin version to increase test coverage. This is done in the PR #178.

5.5 Informational

5.5.1 Number of TODO and question comments remain in the source code

Severity: Informational

Description: A not insignificant number of TODO and question comments remain in the source code. These can lead to uncertainty whether a behavior is by design or not. Combined with a number of Ethereum State Tests cases being disabled, there is a likelihood they mask real issues.

Recommendation: Review all such comments, address and remove them.

Polygon-Hermez: TODO comments and cleaning code has been done in PR #222 and PR #117.

Spearbit: Acknowledged.

5.5.2 Trace log is not saved in case of "register not set to zero" error

Severity: Informational

Description: In case an Ethereum test execution ends with the error **`Error: Program terminated with registers A, D, E, SR, CTX, PC, MAXMEM, zkPC not set to zero`**, a trace file is not created in **`zkevm-proverjs/src/sm/sm_main/logs-full-trace`**, which makes it difficult to debug an error (see some tests that were failing like this in *Test failures in `VMTests/vmPerformance/loopExp` (potential issue in `EXP`)*).

Recommendation: Always create a trace log file if a test's execution ends with an error.

Polygon-Hermez: Fixed in #110.

Spearbit: Acknowledged.

5.5.3 The `identity` precompile inner loop checks unused counters

Severity: Informational

Context: zkevm-rom:indentity.zkasm#24, zkevm-rom:indentity.zkasm#61

Description: The **`identity`** precompile inner loop called **`IDENTITYreturnLoop`** decrements the **`STEPS`**, **`BINARY`** and **`POSEIDON_G`** counters, but those seem not to be used in the loop, given no arithmetic calculation nor hashing is taking place.

Decrementing these are wasteful and can result in less than feasible number of transactions fitting a block.

Recommendation: Do not adjust those counters.

Polygon-Hermez: The suggested approach has been followed regarding checking zk-counters at low function level. Improvement of handling zk-counters has been implemented in PR #223. Furthermore, a safe zk-counter band has been added in order to prevent an non controlled out-of-counters in PR #227.

Spearbit: Acknowledged.

5.5.4 Possible `opPUSH` optimization

Severity: Informational

Context: zkevm-rom:stack-operations.zkasm#L212-L216

Description: In the **`opPUSH`** implementation there is a check if the push data is within the code boundaries:

```
$ => B                  :MLOAD(bytecodeLength)
PC + D => A
$                       :LT,JMPC(opAuxPUSHA2)
PC => A
B - A => D
```

It is not required to perform comparison with 256-bit precision because the values are within 32-bits. So the **`LT`** can be replaced with regular subtraction:

```
$ => B                  :MLOAD(bytecodeLength)
B - PC - D              :JMPN(opAuxPUSHA2)
B - PC => D
```

Moreover, for the pathological case of push data being only partially in the code boundaries, the **`D`** is adjusted to **`B - PC`**. In particular, **`D`** can be 0, so please check if **`readPush`** will handle it correctly.

Finally, it can be noted that in the pathological case the value pushed to the stack is never accessed (as **`PUSH`** is the very last instruction). Therefore, in this case you can always set **`0 => D`**.

Polygon-Hermez: Implemented in PR #242.

Spearbit: Acknowledged.

5.5.5 Testing could benefit from having an option to run individual tests

Severity: Informational

Context: eth-tests-folder.sh#L20

Description: The **eth-tests-folder.sh** script always regenerates inputs, even without the **update** option. This is inconsistent with the **eth-tests.sh** script logic which allows to only run the tests without regeneration.

Recommendation: Change **eth-tests-folder.sh** to make it similar to **eth-tests.sh**:

```
if [ -f "eth-inputs/$group/$folder/info.txt" ] && [ "$1" != "update" ]
then
    echo "Exist"
else
    npx mocha --max-old-space-size=12000 gen-inputs.js --group $group --folder $folder --output
    ↪  eth-inputs
fi
```

Also it would be convenient to have an option to run a single test inside a given folder only.

Polygon-Hermez: README.md was improved to explain the purpose of each script and the ability to run one single test in PR #161.

5.5.6 Not easy to use debug options of test running

Severity: Informational

Context: zkevm-testvectors

Description: The README mentions some extra options, but it's not clear how to activate them: **ethereum-tests#options**. For example, one may want to add **--evm-debug** to a **gen-inputs.js** run.

Also, produced traces are put into undocumented hard-to-find locations:

- ethereumjs traces are saved into **zkevm-testvectors/tools/ethereum-tests/evm-stack-logs** (in case **--evm-debug** is activated),
- zkevm traces are saved into **zkevm-proverjs/src/sm/sm_main/logs-full-trace** (apparently always, no flags needed).

Recommendation: Better documentation on how to collect EVM traces from test execution.

Polygon-Hermez: Better documentation has been added in PR #161.

5.5.7 Possible simplification of SHL

Severity: Informational

Context: zkevm-rom:util.zkasm#L609

Description: The shift left operation is divided into the two cases: zero overflow and nonzero overflow. In both cases, the **ARITH** state machine is called with exactly the same values, with the only difference that in one case **D=0** is used (D is the overflow in the state machine) while in the other **D** is a free input that contain the overflow.

Several calls to **BINARY** and the exponentiation mechanism are used to come up with the fact if the overflow is zero or not. In the end, both cases can just as well use **D** as a free input.

Recommendation: Only use the case **SHLarithBig**:

- return zero if **D** (shift amount) is less than 256,
- compute **B = 2**D**,
- verify if **A * B = <overflow>*2**256 + <result>** using **ARITH**.

Polygon-Hermez: Implemented in PR #242.

Spearbit: Acknowledged.

5.5.8 Random test indices for tests from one file

Severity: Informational

Context: zkevm-testvectors:gen-inputs.js#L185

Description: When there are several tests inside one Ethereum test file, **`gen-inputs.js`** produces separate inputs with the pattern **`<testFileName_index>`**, where **`index`** is seemingly "random". It is the index of how the Javascript engine orders keys inside the object (see gen-inputs.js#L174). It is difficult to find the original test that this generated input corresponds to.

Recommendation: Name generated test input files according to the key name in the original test file, e.g. **`sha3_-d11g0v0_Berlin.json`** instead of **`sha3_4.json`** (not that **`d11`** refers to the index inside **`transaction.data`** array of test filler file).

Polygon-Hermez: We will add better file naming in order to easily identify inputs produced for the **`zkevm-rom`**.

5.5.9 Duplicate constant polynomials in `PaddingKK`

Severity: Informational

Context: zkevm-proverjs:sm_padding_kk.js

Description: **`PaddingKK.crValid`** and **`PaddingKK.r8Valid`** have the same values.

This is different from *Duplicate constant polynomials in `Binary/MemAlign`* as this is not a coincidence between two unrelated state machines, but rather inside the same machine. Both polynomials are referenced in the *Hashing-related State Machines* paper.

`crValid` is introduced:

> we will create a constant column crValid, which will be 1 if and only if we are not at the last block

`r8Valid` however is only mentioned in a lookup page 16.

Recommendation: First, clarify if these two polynomials are supposed to have the same values:

- If so, use a single polynomial and fix the paper accordingly,
- If not, define **`r8Valid`** in the paper and fix the values accordingly.

Polygon-Hermez: We have followed the recommendation. We use a single polynomial (**`r8Valid`**) in PR #107. Documentation is also updated.

Spearbit: Acknowledged.

5.5.10 Testing infrastructure reports OOC execution errors as state root mismatch

Severity: Informational

Context: zkevm-proverjs:sm_main_exec.js#L1852

Description: When running some computation-heavy tests from **`evm_benchmarks`** the observed result output is state root mismatch:

```
Input: 0xPolygonHermez2/zkevm-testvectors/tools/ethereum-tests/eth-inputs/GeneralStateTests/evm_benchma ⌋
↪  rks/JUMPDEST_n0.json
Start executor JS...
Error
Error: Assert Error: newStateRoot does not match
```

However, enabling tracer output shows that actual execution error was OOCS (out of counters):

```
{
  tx_hash: '0xdb9be29bc49dd4615aae90d141869d51b320fb938e896bd3be9c240bd1fa18be',
  rlp_tx: '0xf86380018430000000094be7c43a5800000000000000000000000000000000180808207f3a0c77643128f488ea15b
  ↪  0d6ed338b8f99ad155a0f5a2eb8d5f3efc4c0ac1782a39a043f032bde2bd2b5fbdc154d32886bae6fd6664f
f5f307d9adbf843b60dcd451f',
  type: 0,
  return_value: '',
  gas_left: '29453541',
  gas_used: '546459',
  gas_refunded: '0',
  error: 'OOCS',
  create_address: '',
  state_root: '0x786576b88c34f7812afdb7eb7a9a188948843e8361dd53deb376b895f21927aa',
  logs: [],
  call_trace: {
    context: {
      to: '0xbe7c43a5800000000000000000000000000000001',
      type: 'CALL',
      data: '0x',
      gas: '30000000',
      value: '0',
      batch: '0x00',
      output: '',
      gas_used: '546459',
      execution_time: '',
      old_state_root: '0x786576b88c34f7812afdb7eb7a9a188948843e8361dd53deb376b895f21927aa',
      nonce: 0,
      gasPrice: '1',
      chainId: 1000,
      from: '0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',
      return_value: ''
    },
    steps: [
...
```

Recommendation: In case of OOC, tests shouldn't get to comparing state roots since execution failed. Output error should better reflect what really happened.

Polygon-Hermez: Better terminal output and verbosity has been added in PR #105 and PR #114.

5.5.11 Testing infrastructure onboarding guide is unclear

Severity: Informational

Context: zkevm-testvectors / zkevm-proverjs

Description: The setup of the **`ethereum/tests`** test suite is error prone, especially given the lack of clarity of which branches to use, and the fact that recent changes to some of those branches break runs that would otherwise work until recently.

The updated readme helped, at the very least for validating that we were running the test suite correctly, albeit with some wrong branches occasionally.

Here's the summary of my (currently working) setup:

- **`zkevm-testvectors`**: **`71926534`** (currently **`develop`**)
- **`zkevm-rom`**: **`v0.5.2.0`**
- **`zkevm-proverjs`**: **`1252c73`** (as specified in the new readme)

The main hurdle to get this working though, was the fact that **`zkevm-proverjs`** generates a file at its root, **`cache-main-pil.json`**. When changing branches on this repo, we need to manually delete this file or risk it being

incompatible in the new branch.

This behavior is of course not noticed when cloning the repositories for the first time, but seems to be undocumented and causing problems for anyone switching branches on an existing setup.

Polygon-Hermez: Improved testing infrastructure scripts in PR #161.

5.5.12 `opSDIV` only needs 1 call to XOR in binary SM instead of 2

Severity: Informational

Context: zkevm-rom:arithmetic.zkasm#L143

Description: **`SDIV`** can be implemented with fewer calls to the binary state machine.

`opSDIV` calls **`abs`** on its arguments to get the absolute value and the sign, respectively. If the signs of the arguments are the same, it can use unsigned division on the absolute values. If the signs are different, the result is the negated division of the absolute values.

It computes "are the signs different" in a complicated way by using **`XOR(XOR(sgnA, sgnB), 1)`**. The exact same result can be obtained by using **`EQ(sgnA, sgnB)`**.

Potentially there may be an easier way to compute this value that does not need a state machine at all.

Recommendation: Replace two calls to **`XOR`** by one call to **`EQ`**.

Polygon-Hermez: Optimized in PR #205.

5.5.13 Invalid comment in `opNOT`

Severity: Informational

Context: zkevm-rom:comparison.zkasm#L311

Description: The comment in **`opNOT`** says **`2**226`** but it should be **`2**256`**.

Recommendation: Change comment to **`2**256`**.

Polygon-Hermez: Fixed in PR #224.

5.5.14 Test folder is skipped if input generation runs out of memory for one test file in it

Severity: Informational

Context: eth-tests-get-info.sh#L29

Description: At the end of the test execution run we are getting the following errors:

```
./eth-tests-get-info.sh: line 36: let: err_gen_perc=(100*0+0/2)/0: division by 0 (error token is "0")
./eth-tests-get-info.sh: line 37: let: err_exec_perc=(100*0+0/2)/0: division by 0 (error token is "0")
./eth-tests-get-info.sh: line 38: let: ok_perc=(100*0+0/2)/0: division by 0 (error token is "0")
```

It seems likely that some of the input generations ran out of memory. This part of the log points to it:

```
Test name:  Return50000_2_0.json
WRITE: /0xPolygonHermez/zkevm-testvectors/tools/ethereum-tests/eth-inputs/GeneralStateTests/stQuadratic⌋
↪  ComplexityTest/Return50000_2_0.json

Test name:  Return50000_2_1.json
Killed
```

In the end, the **`stQuadraticComplexity`** folder does not have an **`info.txt`** summary file, and then **`eth-tests-get-info.sh`** fails to find the test.

This probably affects final numbers for test statistics for passed / failed / generation errors. OOM error is buried in the log and is hard to notice that some tests were in fact skipped.

Recommendation: Consider missing **`info.txt`** in the folder as test generation error, that is easy to notice

Polygon-Hermez: Some tests require high-end dedicated hardware to be run. Concretely, **`Return50000_2_-0.json`** is marked as ignored since it triggers an out-of-counters steps on the zkevm-rom after run it in a large machine. We are currently moving all the test triggering an out-of-counters errors into passed by running them without zk-counters restrictions.

5.5.15 Undocumented tests infrastructure requirement

Severity: Informational

Context: eth-tests.sh#L55

Description: Running ethereum tests with **`eth-tests.sh`** script requires **`zkevm-rom`** repo to be cloned and **`rom.json`** file built in it, this is not very obvious when setting up the tests for the first time.

`zkevm-proverjs` repo is required as well.

Recommendation: Either document that clone of **`zkevm-proverjs`**, **`zkevm-rom`**, **`npm install`** and **`npm run build`** are required to run the tests, or include this in setup scripts.

Polygon-Hermez: Documentation has been improved in PR #161.

5.5.16 Test skipped/ignored if transaction `gaslimit > 30M`

Severity: Informational

Context: zkevm-testvectors:gen-inputs.js#L223-L226

Description: Tests are not attempted if their gas limit is above 30M, even when they would spend less gas. Transaction gas limit in tests might be an arbitrary number and not represent how much gas they really need.

Recommendation: Consider changing the test execution tool to either attempt using a larger limit (80M is a frequent arbitrary limit we have seen) or to try executing large limit tests and see if they work. Alternatively individual test cases can be improved to use a realistic limit.

Polygon-Hermez: Tests considered ignored regarding two topics has been addressed and are executed in the **`zkevm-rom`** to increase test coverage. Those tests are the following ones:

- gasUsed > 30.000.000 gas
  - maximum gas limit that could be incremented is up to **`2**32 - 1`** due to **`zkevm-rom`** handling **`GAS`** register
  - test above **`2**32 - 1 gasLimit`** will be checked and new test will be generated with custom filler modifying the transactions in the Ethereum test-suite
- out of counters
  - zero-knowledge resources are wasted. Hence, test cannot be run in **`zkevm-rom`**
  - a mode without taking into account zk resources has been added and then test could be executed in the **`zkevm-rom`**

5.5.17 Duplicate constant polynomials in `Binary`/`MemAlign`

Severity: Informational

Context: zkevm-proverjs:sm_binary.js, zkevm-proverjs:sm_mem_align.js

Description: **`Binary.P_A`** and **`MemAlign.BYTE2A`** are identical. **`Binary.P_B`** and **`MemAlign.BYTE2B`** are identical.

Recommendation: Two constant polynomials could be saved by extracting these (for example into **`Global`**) and using the same constant polynomials in **`Binary`** and **`MemAlign`**.

Polygon-Hermez: We have followed the recommendation in **`binary.pil#L86-L87`** and **`mem_align.pil#L110-L116`**.

5.5.18 Test generator fails to parse stEOF tests (and other too nested directories)

Severity: Informational

Context: testvectors -> gen-inputs.js

Description: The infrastructure for reading tests from the Ethereum Test Suite loops through each individual JSON across the entire directory tree, in order to generate **`eth-inputs`**. However, the traversal code implicitly assumes a maximum depth of 3, which coincidentally was broken just recently with the new **`stEOF/stEIP3540`** files.

Since **`eth-tests.sh`** doesn't enforce a specific commit hash when fetching the test suite, this can create random new failures when attempting to setup tests for the first time.

Recommendation: Enforce an explicit supported version of the Ethereum test suite, by updating the clone logic on **`eth-tests.sh`** or (perhaps preferably) use a git submodule to express the dependency.

Polygon-Hermez: Freeze a commit to Ethereum test-suite in PR #164 and improve parsing tests in PR #161.

5.5.19 Unnecessary stack overflow check in many opcodes

Severity: Informational

Context: opcodes.zkasm#L42

Description: Many instructions in EVM cannot cause stack overflow. Some of these instruction implementations in zkVM have unnecessary stack overflow check **`%CALLDATA_OFFSET - SP :JMPN(stackOverflow)`**, such as **`opNOT`** and **`opDUPx`**.

Recommendation: Remove the stack overflow check from the listed instructions.

Polygon-Hermez: Fixed in PR #202 and #240.

5.5.20 EIP-3541 should be supported

Severity: Informational

Context: zkevm-rom

Description: The test cases called **`FirstByte`** are disabled (see the exception list on **`develop`** and **`main`**). These check that new contracts cannot be created with code starting with the **`0xEF`** byte.

The lack of support is understandable, given zkEVM supports Berlin, and this change was introduced in the London upgrade. However, this is a future-compatibility change rolled out on a number of networks, and likely it would be prove useful for zkEVM to be prepared. The change is relatively minor.

Recommendation: Support EIP-3541.

Polygon-Hermez: EIP-3541 added in PR #210. Custom test has been added in PR #163.

5.5.21 Repositories with `package.json` use Semver-incompatible versions breaking npm/yarn

Severity: Informational

Context: zkevm-testvectors, zkevm-commonjs, zkevm-contracts, zkevm-rom (possibly others)

Description: Cloning and installing zkevm-testvectors is not currently possible without manual intervention. For example checking out **`v0.4.0.0`** and running **`npm install`** or **`yarn install`** will result in the following:

```
% yarn install
yarn install v1.22.11
warning package.json: License should be a valid SPDX license expression
warning package.json: "dependencies" has dependency "chai" with range "^4.3.6" that collides with a
↪  dependency in "devDependencies" of the same name with version "^4.3.4"
info No lockfile found.
warning @0xpolygonhermez/test-vectors@0.4.0.0: License should be a valid SPDX license expression
warning @0xpolygonhermez/test-vectors@0.4.0.0: "dependencies" has dependency "chai" with range "^4.3.6"
↪  that collides with a dependency in "devDependencies" of the same name with version "^4.3.4"
[1/4]   Resolving packages...
error Can't add "@0xpolygonhermez/zkevm-commonjs": invalid package version "0.4.0.1".
info Visit https://yarnpkg.com/en/docs/cli/install for documentation about this command.
```

The invalid package version refers to the line in zkevm-commonjs/package.json:

```
  "version": "0.4.0.1",
```

The problem is that (recent) npm/yarn expects the version numbers to be Semver-compatible (see this relevant issue).

The current workaround is to create copies of all dependencies and replace the version numbers to only contain three dots (e.g. **`0.4.0`**, **`0.4.1`**, **`0.4.0-patch1`**), and additionally the install command must be run on both **`testvectors`** and the copy of **`commonjs`** with the modification.

Note: chose **`Low Risk`** because this versioning problem prevents the building and use of zkevm on some npm/yarn versions. It does not constitute of a security risk.

Recommendation: Use Semver-compatible versions.

PolygonHermez: Improved documentation in README.md to specify versions and software used.

5.5.22 PIL hello world multiplier does not compile

Severity: Informational

Context: Hello-World-Examples.md

Description: Compiling this file with **`pilcom`** returns an error, apparently because the degree of the constraint is too high (3).

```
namespace Multiplier(%N);
    pol constant SET;
    pol commit freeIn;
    pol commit out;
    out' = SET*freeIn + (1 - SET)*(out * freeIn);
```

```
node src/pil.js multiplier.pil
Error: ERROR multiplier.pil:undefined: Degre too high
```

Recommendation: Either introduce an intermediate polynomial, or change the example to an adder:

```
namespace Adder(%N);
    pol constant SET;
    pol commit freeIn;
    pol commit out;
    out' = SET*freeIn + (1 - SET)*(out + freeIn);
```

6 Coverage

6.1 Testing

The review focused on zkevm-testvectors#v0.5.1.0.

The list of excluded test cases is around 2200 entries long, which can be grouped under a few categories:

- Precompile tests,
- Transaction tests (EIP-2930 access list related),
- Selfdestruct tests,
- Tests failing due to using >30M gas, OOC (out of counters) or other resource exhaustion.

We have investigated a number of precompile tests, because even those supported (i.e. **`ecrecover`**) were failing in a number of them. It turns out test cases often combine multiple precompiles (usually **`sha256`**). We did not attempt to adjust these tests.

Our focus has been investigating the ones failing due to resource limits and more specifically the EVM specific ones. This includes **`MemoryTests`**, **`VMTests`**, **`StackTests`**, but there wasn't enough time to cover every case.

6.2 zkASM

The majority of the review took place under zkevm-rom#v0.5.2.0.

The ROM consists of a number of parts:

- ecrecover,
- opcodes,
- precompiles,
- utilities,
- transaction processing.

Due to the time available we have focused on a number of areas:

- Transaction validation,
- **`flow-control.zkasm`**:
  - Jumps,
  - **`CREATE2`** focusing on Keccak-256,
- **`comparison.zkasm`**: **`LT`**, **`GT`**, **`SLT`**, **`SGT`**, **`EQ`**, **`ISZERO`**, **`AND`**, **`OR`**, **`XOR`**, **`NOT`**, **`BYTE`**, **`SHR`**, **`SHL`**, **`SAR`**,
- **`stack-operations.zkasm`**: **`DUPx`**, **`SWAPx`**, **`POP`**, **`PUSH`**,
- **`utils.zkasm`**: a number of utilities used by the above.

6.3 PIL State Machines

The list below contains all zkEVM state machines written in PIL (under zkevm-proverjs#cc474493 and zkevm-proverjs#ee7f7852) and their coverage duing this engagement:

- **`arith`**, partially covered (all except correctness of curve equations).
- **`binary`**, fully covered.
- **`mem`**, fully covered.
- **`mem_align`**, fully covered.
- **`keccakf`**, partially covered.

- **padding_kk**, partially covered.
- **padding_kkbit**, partially covered.
- **nine2one**, fully covered.
- **padding_pg**, partially covered.
- **poseidong**, partially covered.
- **main/rom**, partially covered.
- **global**, fully covered.
- **storage**, not covered.

7 Appendix

8 Formal Analysis of PIL State Machines

In order to aid the audit, we have developed Pilspector, a PoC tool that can perform static analysis, nondeterminism checks, and proofs of correctness for PIL state machines. The current version of the tool was developed in under two weeks and can still be improved considerably.

We report below on the correctness proofs for various properties on some of the state machines that compose Polygon's zkEVM.

8.1 arith.pil

The state machine in **`arith.pil`** was formalized assuming all polynomials represent integer values. This is justified as long as all values and all arithmetic operations stay within the used field.

The state machine was unrolled for a length of 32 rows starting at row 0. The following properties were proved:

- The value of **`Arith.x1`** in row **`0`** is always the same as in row **`31`**. This is proven in **`arith_check_x1_constant_accross_window.smt2`**.
- If equation 0 is selected and in the first row, the following variables are set: **`Arith.y1 = 1`**, **`Arith.y2 = 0`**, **`Arith.x2 = 0`**, **`Arith.x1 = 0`**, **`Arith.y3 = 10`**, then **`Arith.x1`** has to be equal to **`10`** in the first row. These are proved in **`arith_eq0_b1_c0_d0_e10_implies_a10.smt2`**.

8.2 mem.pil

8.2.1 Read

Reads are implied by $mOp' = 1 \wedge mWr' = 0$. The output values $val'_i, i \in [0, 7]$ depend on whether *lastAccess* = 1, that is, the address access before the requested read.

- Lemma 1: $mOp' = 1 \wedge mWr' = 0 \wedge lastAccess = 1 \implies val'_i = 0$. If *lastAccess* = 1 it is the first time that the newly read address is being accessed, therefore $val'_i = 0$. This is proven in **`mem_read_last_access.smt2`** returning **`unsat`**.
- Lemma 2: $mOp' = 1 \wedge mWr' = 0 \wedge lastAccess = 0 \implies val'_i = val_i$. If *lastAccess* = 0 the read address was either read or written before, therefore $val'_i = val_i$. This is proven in **`mem_read_not_last_access.smt2`** returning **`unsat`**.
- Lemma 3: $mOp' = 1 \wedge mWr' = 0 \implies val'_i$ are deterministic, regardless of the value of *lastAccess*. This is proven in **`mem_reads_are_deterministic.smt2`** returning **`unsat`**.

8.2.2 Write

By design, writes may write any values into val'_i. These values are unconstrained in the **`Mem`** state machine, so we can query for nondeterminism but rather expect that it is possible in the presence of a write.

- Lemma 4: $mWr' \implies val'_i$ are nondeterministic. This is shown in **`mem_writes_are_not_deterministic.smt2`** returning **`sat`**.

Another simple yet useful theorem is the relation between writes and memory operations.

- Lemma 5: $mWr' \implies mOp'$, that is, there can never be a write outside a memory operation. This is shown in **`mem_write_implies_mem_op.smt2`** returning **`unsat`**.

8.2.3 Address and Step

The memory access table is sorted by address, and then by step. This is encoded by requiring the address (**`addr`**) to either increase in the next row, or keep its value and increase the step. This is true for all pairs of rows except the pair (last row, first row).

- Lemma 6: $\mathit{ISNOTLAST} = 1 \implies (\mathit{addr}' > \mathit{addr}) \vee (\mathit{addr}' = \mathit{addr} \wedge \mathit{step}' > \mathit{step})$. This is shown in **mem_isnotlast_implies_inc_addr_step.smt2** returning **unsat**.

The other relation of interest here is between **addr** and **lastAccess**, where the address cannot change in the next row if the current one is not the last access of the address.

- Lemma 7: $\mathit{lastAccess} = 0 \implies \mathit{addr}' = \mathit{addr}$. This is shown in **mem_not_last_access_implies_same_addr_smt2** returning **unsat**.