SPEARBIT

Polygon zkEVM Security Review

January 2023 Engagement

Auditors

Alex Beregszaszi, Lead Security Researcher

Andrei Maiboroda, Lead Security Researcher

Christian Reitwiessner, Lead Security Researcher

Leo Alt, Lead Security Researcher

Paweł Bylica, Lead Security Researcher

Thibaut Schaeffer, Lead Security Researcher

Lucas Vella, Security Researcher

Blockdev, 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 11 days in total, Polygon engaged with Spearbit to review the Polygon zkEVM protocol. In this period of time a total of **30** issues were found.

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

1. Continued manual and tooling aided review of state machines (in PIL).
2. Understanding and analysis of zkASM<>PIL integration (crossing the boundaries).
3. Initial review of the construction of constant polynomials.
4. Continued review of the status of excluded Ethereum state tests and verify the reason for exclusion.
5. Continued manual review of the zkASM implementation of EVM instructions.

Similarly to our previous engagement, our researchers formed two groups:

- one focused on state machines and PIL,
- the other focused on instruction implementations and testing assessment.

We recommend a follow up engagement to increase review coverage.

Summary

Project Name	Polygon
Repository	See section 6
Type of Project	Zero Knowledge EVM, zk-rollup
Audit Timeline	Jan 9th - Jan 23rd

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	3	3	0
Incompatibility	4	2	2
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	21	19	2
Total	30	26	4

5 Findings

5.1 Critical Risk

5.1.1 SENDALL zeroes out balance in case receiver equals sender

Severity: Critical Risk

Context: zkevm-rom:create-terminate-context.zkasm#L1040

Description: The **SENDALL** implementation updates the receiver's balance, then zeroes out the sender's balance without ever checking whether they are the same address. Running **SENDALL** with the receiver being the same as the executing contract results in balance 0.

As noted in *SENDALL is untested in the Ethereum Test Suite*, testing of **SENDALL** is not covered in the Ethereum test suite. To confirm, we ran ethereum/tests#1024, which required a number of changes:

- Disabling skipping **SELFDESTRUCT** tests in test infrastructure at zkevm-testvectors:gen-inputs.js#L406
- Modifying go-ethereum to activate **SENDALL** in Berlin
- Fixing a bug in gas charge in go-ethereum implementation (find modified version at internal repository)
- Modifying the tests tests to be run on Berlin (find modified version at internal repository)

Outcome is that **sendallBasic** passes and **sendallToSelf** fails at generating inputs:

```
Test name:  sendallToSelf.json
AssertionError: expected '0' to equal '1000000000000000000'
    at Context.<anonymous>
    ↪  (/0xPolygonHermez/zkevm-testvectors/tools/ethereum-tests/gen-inputs.js:452:92)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5) {
  showDiff: true,
  actual: '0',
  expected: '1000000000000000000',
  operator: 'strictEqual'
}
```

While **SELFDESTRUCT** uses this edge case for "burning" Ether, the **SENDALL** proposals seems to be not wanting this edge case as evidenced by the tests.

Recommendation: Do not update balance in case the receiver is the same as the executing contract.

Polygon-Hermez:

- Fixed in PR #236.
- Added provided test to run automatically on each ROM change in PR #172.
- Modified **ethereumjs-monorepo** to handle **sendall** to itself properly.

Spearbit: Acknowledged.

5.1.2 `opCALLDATALOAD`/`opCALLDATACOPY` reading position out of calldata bounds

Severity: Critical Risk

Context: zkevm-rom:calldata-returndata-code.zkasm#L24, zkevm-rom:calldata-returndata-code.zkasm#L128

Description: The check for input calldata offset being within calldata bounds is done as `txCalldataLen < offset`. In case `offset == txCalldataLen`, it proceeds to load the memory word at address corresponding to position `calldata[txCalldataLen]` (memory word at address `1024 + txCalldataLen / 32`).

Recommendation: Check `offset < txCalldataLen`.

Polygon-Hermez: Implemented optimization in the PR #242 and added comment regarding out-of-bounds in `calldata-returndata-code.zkasm#L38`.

Spearbit: Acknowledged.

5.1.3 `opJUMP`/`opJUMPI` reading position out of code bounds

Severity: Critical Risk

Context: flow-control.zkasm#L51

Description: The check for jump destination being within code bounds is done as `bytecodeLength < jumpDst`. In case `dst == bytecodeLength` it proceeds to fetch bytecode at position `code[bytecodeLength]`.

Recommendation: Check `dst < bytecodeLength` instead, then `JMPNC(invalidJump)`.

Polygon-Hermez: Fixed in PR #226.

Spearbit: Acknowledged.

5.2 Incompatibility

5.2.1 Missing support for certain precompiles can cause usability and security risks

Severity: Incompatibility

Context: selector.zkasm

Description: The Berlin protocol release contains 9 precompiles (`ecrecover`, `sha256`, `ripemd160`, `identity`, `modexp`, `ecadd`, `ecmul`, `ecpairing`, `blake2f`), but zkEVM only implements 2 of them (`ecrecover` and `identity`).

Regarding the unsupported ones:

- `ripemd160` and `blake2f` are rarely used,
- `modexp` has a common use case of RSA verification, but has also been used in general purpose bignumber libraries,
- `ecadd`/`ecmul`/`ecpairing` is used for various zero knowledge applications, examples include bridging, private transactions, and gaming applications.

A prominent example for RSA is ENS' dnssec-oracle, which is used to validate ownership of DNSSEC enabled domains (such as .xyz, .tech, and a large number of others). Another interesting example is a VDF verification (1, 2), which has been seen in some experimental contracts.

The lack of these can predominantly manifest in two ways:

1) **Usability problems.** Certain projects won't be able to deploy on zkEVM, which may have a larger impact on adoption and cross-chain compatibility.
2) **Security problems.** In many cases it would be possible to deploy such projects, but certain features would become broken. it may be possible in that case for value to become locked up in such contracts.

Recommendation: Prominently note the differences in documentation. Consider implementing **sha256**, **modexp** and the bn256 precompiles.

Polygon-Hermez: We have decided to revert any call to a precompile that is not supported for security reasons.

It is implemented in **selector.zkasm#L16** and in **revert-precompiled.zkasm**.

5.2.2 BASEFEE instruction returns fixed 0 value

Severity: Incompatibility

Context: zkevm-rom:block.zkasm#L180

Description: The **BASEFEE** instruction is not part of Berlin, but was introduced with London as part of the EIP-1559 rollout. There is no explicit value range defined, nor is the value 0 explicitly mentioned in any form.

Concern 1. EIP-1559 defines **base_fee_per_gas** as a block header element and the value of this field has to follow a set of rules:

```
INITIAL_BASE_FEE = 1000000000
BASE_FEE_MAX_CHANGE_DENOMINATOR = 8
ELASTICITY_MULTIPLIER = 2

...
parent_gas_target = self.parent(block).gas_limit // ELASTICITY_MULTIPLIER
parent_gas_limit = self.parent(block).gas_limit
# on the fork block, don't account for the ELASTICITY_MULTIPLIER to avoid
# unduly halving the gas target.
if INITIAL_FORK_BLOCK_NUMBER == block.number:
    parent_gas_target = self.parent(block).gas_limit
    parent_gas_limit = self.parent(block).gas_limit * ELASTICITY_MULTIPLIER

parent_base_fee_per_gas = self.parent(block).base_fee_per_gas
parent_gas_used = self.parent(block).gas_used
...
# check if the base fee is correct
if INITIAL_FORK_BLOCK_NUMBER == block.number:
    expected_base_fee_per_gas = INITIAL_BASE_FEE
elif parent_gas_used == parent_gas_target:
    expected_base_fee_per_gas = parent_base_fee_per_gas
elif parent_gas_used > parent_gas_target:
    gas_used_delta = parent_gas_used - parent_gas_target
    base_fee_per_gas_delta = max(parent_base_fee_per_gas * gas_used_delta // parent_gas_target //
    ↪  BASE_FEE_MAX_CHANGE_DENOMINATOR, 1)
    expected_base_fee_per_gas = parent_base_fee_per_gas + base_fee_per_gas_delta
else:
    gas_used_delta = parent_gas_target - parent_gas_used
    base_fee_per_gas_delta = parent_base_fee_per_gas * gas_used_delta // parent_gas_target //
    ↪  BASE_FEE_MAX_CHANGE_DENOMINATOR
    expected_base_fee_per_gas = parent_base_fee_per_gas - base_fee_per_gas_delta
assert expected_base_fee_per_gas == block.base_fee_per_gas, 'invalid block: base fee not correct'
```

Generally the lowest value used across the ecosystem (for testing and tooling) is 7 wei.

For the Ethereum Mainnet, the minimum possible value for the **basefee** is 7 wei. But it also depends on the initial value for the fork block (**INITIAL_BASE_FEE**). Here's a proof of this:

Let us define the following variables

1. b_n denotes the base-fee of the block n.
2. g_n denotes the total gas used by the block n.
3. t_n denotes the target gas of the block n.

For simplicity, assume that block 0 is the EIP-1559 Hardfork block where $b_0 = 10$ gwei. Now, given a block n, the basefee for block $n+1$ is defined by

$$b_{n+1} = \begin{cases} b_n & \text{if } g_n = t_n \\ b_n + \max\left(\left\lfloor \frac{b_n \cdot (g_n - t_n)}{8 \cdot t_n} \right\rfloor, 1\right) & \text{if } g_n > t_n \\ b_n - \left\lfloor \frac{b_n \cdot (t_n - g_n)}{8 \cdot t_n} \right\rfloor & \\ & \text{if } g_n < t_n \end{cases}$$

We can prove inductively that $b_n \geq 7$ wei for all $n \in \mathbb{N}$. For the base case, $n = 0$, $b_0 = 10$ gwei ≥ 7 wei. Now, assuming that $b_n \geq 7$, we can show that $b_{n+1} \geq 7$. For the first case, this is trivial as $b_{n+1} = b_n \geq 7$. For the second case, since $b_{n+1} > b_n \geq 7$, this is also true.

For the last case, i.e., when $g_n < t_n$, the minimum value for the expression $b_n - \left\lfloor \frac{b_n \cdot (t_n - g_n)}{8 \cdot t_n} \right\rfloor$ happens when $g_n = 0$, where it takes the value $b_n - \lfloor \frac{b_n}{8} \rfloor$. When $b_n = 7$, this evaluates to 7. Now, assume that $b_n \geq 8$, then $b_n - \lfloor \frac{b_n}{8} \rfloor \geq b_n - \frac{b_n}{8} = b_n \cdot \frac{7}{8} \geq 7$.

While, a **basefee** of 7 wei is a good default value, even with the fixed 0 as **basefee**, the zkEVM can always define a fork block for EIP-1559 and set $b_{\text{fork-block}}$ to a constant higher than 7 wei to kick-start a proper EIP-1559 mechanism.

Note: once the **basefee** is at least 7 wei, it can never get below 7 wei. If transaction processing follows EIP-1559, then after 7 blocks with $b_n > t_n$, then **basefee** reaches 7 wei.

Note: one interesting consequence of EIP-1559 was that it prevented transactions that had 0 as the **gasprice**. It's probably worth checking what happens when a sequencer includes a transaction with 0 as its **gasprice**.

Note: If the **basefee** is a fixed 0 value, then the EIP-1559 mechanism becomes inconsistent if the case $b_n > t_n$ can be reached, i.e., if the gas used by a block is above half of the block gas limit.

Concern 2. The Ethereum Test Suite does not contain any test with **currentBaseFee: 0**, and so this cases' interaction with other parts is not covered.

Concern 3. EIP-3198 introducing the instruction itself presents a few motivations:

2. Gas futures can be implemented based on it. This would be more precise than gastokens.
3. Improve the security for state channels, plasma, optirolls and other fraud proof driven solutions. Having the **BASEFEE** as an input allows you to lengthen the challenge period automatically if you see that the **BASEFEE** is high.

These two cases cannot be supported with a constant 0 value, and so projects depending on this could fail (or lock up) in various manners on zkEVM.

Concern 4. Lastly, some tools (such as hardhat) have experienced issues the value 0 in their gas estimation subsystem, causing testing or deployment code to fail.

Recommendation: Document this deviation from Mainnet/Berlin. Consider a non-zero value or removal of this instruction.

Polygon-Hermez: Taking into account all your considerations and comments, we have decided to remove the **BASEFEE** opcode from the current zkEVM version for two main reasons:

- **security considerations**: this opcode is not implemented in Berlin hardfork. Contracts that implemented this opcode would be supported but an unexpected value from the opcode will be taken since it will be hardcoded. This may decrease security guarantees of contacts using this opcode,
- **consistency:** addition/removal of this opcode does not imply restriction in terms of performance on zkEVM. Therefore, as Berlin hardfork is implemented and EIP-1559 is not included, it does not add any advantage but supporting contracts using this opcode which may result in security issues (see previous point).

5.2.3 `SENDALL` is untested in the Ethereum Test Suite

Severity: Incompatibility

Context: zkevm-rom:create-terminate-context.zkasm#L979

Description: The **`SENDALL`** instruction is not covered in the Ethereum Test Suite, because it is a work in progress proposal (EIP-4758). A PR exists from last year, but its conformance to the EIP is unclear.

This issue is connected to *`SELFDESTRUCT` instruction is replaced with `SENDALL`*.

Recommendation: Implement tests with correct coverage.

Polygon-Hermez: Noted.

5.2.4 `SELFDESTRUCT` instruction is replaced with `SENDALL`

Severity: Incompatibility

Context: zkevm-rom:create-terminate-context.zkasm#L979

Description: The **`SELFDESTRUCT`** instruction is not conforming to the specification of Berlin. Instead, it is replaced with the (partial) semantics of EIP-4758: **`SENDALL`**, where the entire account balance is still transferred, but the account *is not removed*. Additionally the gas refund is removed. The EIP however is unclear whether execution terminates or not.

This is a significant difference, and smart contract authors must be aware.

Also note, that while this change has been proposed and discussed for quite some time on Mainnet, there is no current consensus about how to implement it.

Recommendation: Document this deviation from Mainnet/Berlin.

Polygon-Hermez:

- The current implementation halts the execution as **`SELFDESTRUCT`** does. We are aware that this is not specified in EIP-4758 spec. Our decision was to follow what the **`SELFDESTRUCT`** did, halt transaction execution.
- This behavior (halt execution), the removal of the **`SELFDESTRUCT`** and the instruction of the **`SENDALL`** will be documented properly.

5.3 Low Risk

5.3.1 Possible use of an incomplete block in `padding_pg`

Severity: Low Risk

Context: zkevm-proverjs:padding_pg.pil

Description: The **`padding_kk`** state machine features blocks on 136 rows. Since 136 does not divide **`N`**, there is an incomplete block at the bottom of the table. The **`padding_kk`** state machine disallows access to this last block to "avoid problems" (sm-hash.pdf, page 11).

Similarly, the **`padding_pg`** state machine features blocks of 56 rows. Since 56 does not divide **`N`** either, there is an incomplete block here as well. However, **`padding_pg`** does not seem to disallow access to this block.

Related internal discussion.

Recommendation: After analysis, a concrete exploit has not been found. However, to be on the safe side, we recommend disallowing access to the last block of **`padding_pg`** the same way it is done in **`padding_kk`**.

Polygon-Hermez: We have followed the recommendation in PR #127.

5.3.2 PC should be constrained to zero initially

Severity: Low Risk

Context: zkevm-proverjs:main.pil

Description: The PC values (for the main state machine as well as the storage state machine) are constrained only through the plookup and the constraint that encodes control-flow. There is no constraint that enforces the program to start at the entry point.

Recommendation: It is not totally clear which problems this could lead to (since the execution is cyclic it might not matter where to start), but we recommend to constrain **`Main.zkPC`** and **`Storage.pc`** to zero initially: **`Global.L1 * Main.zkPC = 0`**, **`Global.L1 * Storage.pc = 0`**

Polygon-Hermez: The recommendation was followed in PR #121.

5.4 Informational

5.4.1 `crOffset` should uniquely determine `crF_i` registers

Severity: Informational

Context: padding_kk.pil#L113-L116

Description: **`crOffset`** and **`crF_i`** registers are constrained via plookup arguments. However, the way they are described in Table 7 in the documentation is incorrect.

`crOffset` should uniquely determine **`crF_i`** registers but that's not the case from Table 7. To observe that, note that when **`crLatch`** is 1, **`crOffset`** is 0. However, **`crF_i`** register values are different in both these cases from the table above.

From plookup argument, when **`crOffset==0`**, **`crF_0==1`** and the rest of **`crF_i`** registers should be 0. That's also how it's implemented in PIL.

Recommendation: Update the reference PDF to document the implementation.

Polygon-Hermez: This document was incorrect and has been updated. The offset decreasing and factors must be in inverse order.

5.4.2 `resultRd` is potentially underconstrained in `mem_align.pil`

Severity: Informational

Context: zkevm-proverjs:mem_align.pil#L95

Description: The three polynomials **`resultWr8`**, **`resultWr256`** and **`resultRd`** are constrained to 0 when **`RESET`** is 0. **`resultWr8`** and **`resultWr256`** are also constrained to 0 when **`wr8`** and **`wr256`** are 0, respectively. The same logic could potentially apply to **`resultRd`**, where it would be constrained to 0 when either **`wr8`** or **`wr256`** is 1.

Recommendation: Add the constraint **`(wr8 + wr256) * resultRd = 0`**.

Polygon-Hermez: The explicit constraint is not necessary for soundness since it is guaranteed by permutation checks. The recommendation was nonetheless followed to improve clarity in PR #129.

5.4.3 Redundant constraint in `mem_align.pil`

Severity: Informational

Context: zkevm-proverjs:mem_align.pil#L104

Description: The constraint `wr8 * wr256 = 0;` is redundant, considering `wr8'` and `wr256'` are part of the same plookup in line 111, and the corresponding constants `WR256` and `WR8` ensure that at most one of them equals 1 at a given time.

Recommendation: Remove the constraint.

Polygon-Hermez: The recommendation was followed in PR #129.

5.4.4 `Binary.lCout` is redundant

Severity: Informational

Context: zkevm-proverjs:binary.pil

Description: `lCout` appears in two places.

In `binary.pil`:

```
    lCout' = cOut;
```

and in `main.pil`:

```
    bin {
        binOpcode,
        A0, A1, A2, A3, A4, A5, A6, A7,
        B0, B1, B2, B3, B4, B5, B6, B7,
        op0, op1, op2, op3, op4, op5, op6, op7,
        carry
    } is
    Binary.resultBinOp {
        Binary.lOpcode,
        Binary.a[0], Binary.a[1], Binary.a[2], Binary.a[3], Binary.a[4], Binary.a[5], Binary.a[6],
        ↪ Binary.a[7],
        Binary.b[0], Binary.b[1], Binary.b[2], Binary.b[3], Binary.b[4], Binary.b[5], Binary.b[6],
        ↪ Binary.b[7],
        Binary.c[0], Binary.c[1], Binary.c[2], Binary.c[3], Binary.c[4], Binary.c[5], Binary.c[6],
        ↪ Binary.c[7],
        Binary.lCout
    };
```

This is equivalent to removing `lCout` and changing the permutation check to

```
    bin' {
        binOpcode',
        A0', A1', A2', A3', A4', A5', A6', A7',
        B0‘, B1‘, B2', B3', B4', B5', B6', B7',
        op0', op1', op2', op3', op4', op5', op6', op7',
        carry'
    } is
    Binary.resultBinOp' {
        Binary.lOpcode',
        Binary.a[0]', Binary.a[1], Binary.a[2]', Binary.a[3]', Binary.a[4]', Binary.a[5]',
        ↪  Binary.a[6]', Binary.a[7]',
        Binary.b[0]', Binary.b[1]', Binary.b[2]', Binary.b[3]', Binary.b[4]', Binary.b[5]',
        ↪  Binary.b[6]', Binary.b[7]',
        Binary.c[0]', Binary.c[1]', Binary.c[2]', Binary.c[3]', Binary.c[4]', Binary.c[5]',
        ↪  Binary.c[6]', Binary.c[7]',
        Binary.cOut
    }
```

Recommendation: We do not necessarily recommend this change due to a loss of clarity. Instead, such optimizations, if considered, should be taken care of by the tooling.

Polygon-Hermez: We acknowledge the analysis and do not apply the change.

5.4.5 ModExp precompile is disabled, but source is present

Severity: Informational

Context: selector.zkasm#L16, modexp.zkasm

Description: The ModExp precompile contract has a (partial) implementation, which contains **TODO**s and likely is incomplete. In the current tag for review this precompile has been disabled in **selectorPrecompiled** jump-table.

Recommendation: Remove the incomplete source code from the code base to avoid confusion.

Polygon-Hermez: Clean up performed in PR #228.

5.4.6 Testing report

Severity: Informational

This is the summary of all tests in ethereum state test format that we tried to run.

Ethereum State Tests We enabled a number of state tests that are currently disabled in Polygon zkEVM's test suite. To enable them, it was necessary to either reduce the transaction gas limit, so that it's below 30M gas, and/or enable Berlin hard fork in the tests.

See full report at "Ethereum Tests" page here.

All test modifications are available at the internal repo.

ECRecover tests Some tests for the ECRecover precompile were not being run for Berlin fork. After enabling missing tests, all of them pass.

See full report at the "ECRecover" page here.

Modified tests can be found at the internal repo.

evm-benchmarks Ipsilon's EVM benchmark suite is already in Ethereum State test format. We needed to generate them also in Blockchain test format and enable Berlin fork to test with zkEVM.

Most of the benchmarks end with correct results. Some result in OOC errors, or running out of memory on the testing machine. See full report at the "evm-benchmarks" page here.

Find full logs here.

Generated blockchain tests + fillers are available at the internal repo.

New arithmetic test The test from PR #1144 required to change hardfork to Berlin and passed successfully after that.

Find the modified test at the internal repo.

5.4.7 Incorrect comment on `mem_align` constant definition

Severity: Informational

Context: zkevm-proverjs:mem_align.pil, zkevm-proverjs:sm_mem_align.js

Description: The **`FACTOR`** constants are documented to have a period of 64 rows, the second 32 of which are 0.

However in the executor, this second block of zeroes does not exist, and the **`FACTOR`** are simply the usual diagonal shifts, repeated every 32 rows.

Recommendation: If the constant generation is correct, which is our assumption, fix the comment in the **`mem_align.pil`** file.

Polygon-Hermez: The recommendation was followed in PR #124.

5.4.8 Some intermediate polynomials can be inlined

Severity: Informational

Context: zkevm-proverjs:pil

Description: A number of intermediate polynomials are linear.

Based on static analysis:

```
pol Arith.eq0_31 = Arith.y2[15]
pol Arith.eq1_31 = -262140
pol Arith.eq2_31 = -262140
pol Arith.eq3_31 = -262140
pol Arith.eq4_31 = -262140
pol Binary.RESET = (Global.CLK32[0] + Global.CLK32[16])
pol Mem.INCS = (Global.STEP + 1)
pol Mem.ISNOTLAST = (1 - Global.LLAST)
pol MemAlign.RESET = Global.CLK32[0]
pol PoseidonG.a0 = (PoseidonG.in0 + PoseidonG.C[0])
pol PoseidonG.a1 = (PoseidonG.in1 + PoseidonG.C[1])
pol PoseidonG.a10 = (PoseidonG.cap2 + PoseidonG.C[10])
pol PoseidonG.a11 = (PoseidonG.cap3 + PoseidonG.C[11])
pol PoseidonG.a2 = (PoseidonG.in2 + PoseidonG.C[2])
pol PoseidonG.a3 = (PoseidonG.in3 + PoseidonG.C[3])
pol PoseidonG.a4 = (PoseidonG.in4 + PoseidonG.C[4])
pol PoseidonG.a5 = (PoseidonG.in5 + PoseidonG.C[5])
pol PoseidonG.a6 = (PoseidonG.in6 + PoseidonG.C[6])
pol PoseidonG.a7 = (PoseidonG.in7 + PoseidonG.C[7])
pol PoseidonG.a8 = (PoseidonG.hashType + PoseidonG.C[8])
pol PoseidonG.a9 = (PoseidonG.cap1 + PoseidonG.C[9])
```

Recommendation: Introduce a way to define them as aliases, or inline them during PIL compilation.

5.4.9 Language and tooling could be improved

Severity: Informational

Context: everywhere

Description: The fact that information / code pertaining to one aspect of the zkEVM is often spread across multiple files makes it really hard to review the code. There are other language-based features that could help, for example, constant definitions could be part of the PIL file directly and polynomials could be typed (code history suggests this may have been partially implemented in the past).

The execution semantics and the verification semantics could be in once place. Currently, the executor fills the committed polynomials in a very similar way to how they are verified through constraints in the PIL file.

Recommendation: This is an informational suggestion, since big changes like this are unrealistic to be considered and completed until launch.

Polygon-Hermez: Related to types of polynomials, they are removed because polynomials internally are finite fields (goldilocks). To define one specific type or range, you must define specific constraints or plookups.

5.4.10 Remove `firstHash` in `padding_kk`

Severity: Informational

Context: zkevm-proverjs:padding_kk.pil#L38

Description: The intermediate polynomial **`firstHash`** can be expressed using the committed polynomial **`lastHash`**.

The only uses of **`firstHash`** are the following:

```
pol commit firstHash;
firstHash' = lastHash;
[...]
len * firstHash = rem * firstHash;
```

This is equivalent to:

```
len' * lastHash = rem' * lastHash;
```

Recommendation: The current implementation is a bit clearer than the proposed equivalent thanks to the introduction of a different name **`firstHash`**. However, clarity could be achieved by letting **`firstHash`** be an alias for **`lastHash'`** rather than an intermediate polynomial. This could be done either by having an alias syntax in PIL, or with an optimisation step which tries to inline intermediate polynomials. In the short term, using the proposed equivalent is suggested.

Polygon-Hermez: The recommendation was followed in the PR #123.

5.4.11 Optimize `opAuxPUSHB`

Severity: Informational

Context: stack-operations.zkasm#L169

Description: Two steps can be combined into a single one (as seen in many other places in the code).

Split:

```
    $ => A                          :MLOAD(isCreate)
    0 - A                           :JMPN(opAuxPUSHBcreate)
    ; set bytes length to read to C
    D - 1 => C
    0 => A
```

Combined:

```
    $ => A                          :MLOAD(isCreate), JMPNZ(opAuxPUSHBcreate)
    ; set bytes length to read to C
    D - 1 => C
```

Recommendation: Consider the optimization above.

Polygon-Hermez: The suggestion was implemented in #228.

5.4.12 Memory expansion gas check is split into two checks

Severity: Informational

Context: zkevm-rom:utils.zkasm#L451

Description: The first out-of-gas check is: **`GAS + 3 * old_size + old_size * old_size / 512 < 3 * new_size`** => **`GAS < 3 * new_size - 3*old_size - old_size * old_size / 512`**

The second check is: **`GAS + 3 * old_size + old_size * old_size / 512 - 3 * new_size < new_size * new_size / 512`** => **`GAS < 3 * new_size + new_size * new_size / 512 - 3 * old_size - old_size * old_size / 512`**

Recommendation: This looks to produce the correct result, but can be done with just the second check alone.

Polygon-Hermez: A refactor of the **`saveMem`** function was added in PR #221.

5.4.13 Current memory size is stored as last requested max size, not rounded up to multiple of 32

Severity: Informational

Context: zkevm-rom:utils.zkasm#L440

Description: **`memLength`** variable stores the exact size required at expansion, not rounded up to a multiple of the word size. If a higher address accessed is within the current highest word, it still calculates the memory expansion cost (resulting in 0, as the number of words doesn't change). Also the **`MSIZE`** implementation needs to calculate rounding up to word size from **`memLength`**.

Recommendation: Possible optimization: store the size rounded up to a word size multiple in **`memLength`** (i.e. the size that **`MSIZE`** would return). Then memory expansion will compare the exact requested size against **`MSIZE`** and calculate the expansion cost only if the number of allocated words increases.

In other words you can see this as memory expansion always expanding to a multiple of 32, not to the exact size requested.

Polygon-Hermez: Implemented in PR #238.

5.4.14 `JMP` to a helper used instead of `CALL`

Severity: Informational

Context: zkevm-rom:utils.zkasm#L303, zkevm-rom:utils.zkasm#L60, zkevm-rom:utils.zkasm#L71 and others.

Description: In many places the pattern **`zkPC+1 => RR    :JMP(helper)`** is used.

Recommendation: This could be simplified to using **`CALL(helper)`**. This improves clarity.

Polygon-Hermez: Implemented in PR #226. Deeper analysis will be done to optimize this pattern across all the code.

5.4.15 Potential optimization of `MSTORE`

Severity: Informational

Context: utils.zkasm#L220

Description: `MSTOREX2` utility uses `SHLArith` and `SHRArith` to combine values currently stored in two adjacent memory slots with the new value that will be written across two slots.

Recommendation: This potentially can be optimized by using `MEM_ALIGN_WR`, as `memAlignOptionMSTORE` utility is already doing.

Polygon-Hermez: The suggested optimization has been discussed and studied. It will be taken into account in future versions.

5.4.16 Duplicate key and value in lookup argument

Severity: Informational

Context: zkevm-proverjs:main.pil#L526

Description: The `inHASHPOS` column is present twice in the key of the lookup argument, and the associated target column `Rom.inHASHPOS` is present twice in the value.

```
(x in y) => ({x, x} in {y, y})
```

Recommendation: Deduplicate the keys and the values.

Polygon-Hermez: The recommendation was followed in PR #121.

5.4.17 Unnecessary storing register value in `MSTORE32`

Severity: Informational

Context: zkevm-rom:utils.zkasm#L182

Description: In `MSTORE32` it is not necessary to store the previous value of register `E`, because it is used as input/output register.

Recommendation: Remove the `E :MSTORE(tmpVarE)` statement.

Polygon-Hermez: Recommendation implemented in PR #226 and improved handling of `tmpVars` on `utils` function in PR #212.

5.4.18 Jump destination validity check can be optimized

Severity: Informational

Context: zkevm-rom:flow-control.zkasm#L91

Description: The validity check of jump destinations in initcode involves loading 1 byte with `MLOADX`, right-shifting 31 bytes using `SHRarith`, and comparing against `0x5b`.

Recommendation: A potential optimization is to skip shifting by comparing against `0x5b00..00` (32 bytes). This way the arithmetic counters will not be wasted on `SHRarith`.

Polygon-Hermez: Implemented in PR #226.

5.4.19 `@zk-counters` comments disagree with code

Severity: Informational

Context: zkevm-rom:flow-control.zkasm#L5, zkevm-rom:storage-memory.zkasm#L8 and zkevm-rom:storage-memory.zkasm#L43, etc.

Description: Comments about required counter numbers are out of sync with what the code actually checks in many places.

Recommendation: Either update comments or code.

Polygon-Hermez: Comments regarding **`zk-counters`** have been updated here. Besides, specific tooling has been build to check automatically changes on **`zk-counters`** at each function so any change can be noticed if any code change adds/removes counters.

Spearbit: We warmly welcome the creation of tools.

5.4.20 Typo in `main_executor.js`

Severity: Informational

Context: zkevm-proverjs:main_executor.js#L60

Description: Typo in the word "permited".

Recommendation: Should be spelled "permitted".

Polygon-Hermez: Fixed in PR #119.

5.4.21 Typo in `padding_kkbit.pil` and in `sm-hash` paper

Severity: Informational

Context: zkevm-proverjs:padding_kkbit.pil#L21

Description: This should be **`9*136 = 1224`**.

Recommendation: Change here as well as in the **`sm-hash`** paper.

Polygon-Hermez: Fixed in PR #119.

6 Coverage

6.1 Testing

The review focused on zkevm-testvectors#v0.6.0.0-rc1.

The list of excluded test cases is around 2300 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#audit.1.

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:

- **`arithmetic.zkasm`**: **`ADD`**, **`MUL`**, **`SUB`**, **`DIV`**, **`SDIV`**, **`MOD`**, **`SMOD`**, **`ADDMOD`**, **`MULMOD`**, **`EXP`**, **`SIGNEXTEND`**,
- **`calldata-returndata-code.zkasm`**: **`CALLDATALOAD`**, **`CALLDATASIZE`**, **`CALLDATACOPY`**,
- **`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`**,
- **`storage-memory.zkasm`**: **`MLOAD`**, **`MSTORE`**, **`MSTORE8`**, **`MSIZE`**, **`SLOAD`**,
- **`utils.zkasm`**: a number of utilities used by the above.

We did not cover certain parts:

- secp256k1 implementation,
- several parts of transaction processing,
- certain arithmetic and storage opcodes,
- and various cases, including dependency order of instruction implementations (especially the case of "temporary variables" used for saving values).

6.3 PIL State Machines

The list below contains all zkEVM state machines written in PIL (under zkevm-proverjs#audit.1) 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`**, partially covered.