Polygon zkEVM Cryptography Security Review

Auditors

Wilson Nguyen, Lead Security Researcher

Nirvan Tyagi, Lead Security Researcher

Thibaut Schaeffer, Lead 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 cutting-edge scaling solution that harnesses the power of zero-knowledge proofs in order to reduce transaction costs and massively increase throughput, all while inheriting the security of Ethereum.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Polygon zkEVM according to the specific commit. Any modifications to the code will require a new security review.

3 Executive Summary

Over the course of 2 weeks in total, Polygon Hermez engaged with Spearbit to review the Cryptography parts of the Polygon zkEVM protocol. This review is part of the larger zkEVM engagement.

The team conducted an analysis of the cryptographic proof system used by the zkEVM system. Particular attention was paid to

- the novel PIL (polynomial interpolation language) arguments including lookups, multiset equality, and connection, and
- the STARK recursion pipeline in which proofs are compressed and composed recursively to achieve efficient verification.

The team reviewed the provided white papers outlining the cyrptographic proof system and followed up by reviewing the implementation to gain confidence that the protocol was implemented faithfully with respect to the written description. Overall, no major soundness issues were discovered in either the cryptography or implementation review. A number of smaller editorial, informational, and optimization issues were surfaced, however none require immediate action from the Polygon team.

Across the paper and the code there are a number of findings: 5 with minor severity, 4 informational, 2 inconsistencies, and 3 optimization suggestions. It must be noted that due to time constraints the review of the code is incomplete and further investigation is recommended (see section 4.2.1 for further details).

Summary

Project Name	Polygon zkEVM
Repository	See Below
Type of Project	Cryptography Review
Audit Timeline	March 8th - March 20th

- PIL
  - *Polynomial Identity Language (PIL): A Machine Description Language for Verifiable Computation*
  - v1.0, February 13 2023
- eSTARK
  - *eSTARK: Extending the STARK Protocol with Arguments*

  - v1.0, February 10, 2023
  - v1.2, March 15 2023
- Recursion
  - *Recursion, aggregation, and composition of proofs*
  - v1.0 February 13 2023
- fflonk
  - *fflonk: a Fast-Fourier inspired verifier efficient verson of PlonK*
  - ePrint Archive
  - Polygon Documentation (February 20 2023)
- Recursion pipeline
  - Repository: zkevm-proverjs at branch **`v0.8.0.0-rc.2-forkid.2`**
  - Build documentation
  - The generated STARK verifiers in the recursion pipeline for c12a, rec1, rec2, recf, and fflonk.
- STARK prover and verifier for PIL
  - Repository: pil-stark
  - Files: stark_gen, stark_setup, stark_verify, starkinfo, starkinfo_cp_prover, starkinfo_cp_verifier, starkinfo_fri_prover, starkinfo_fri_verifier, starkinfo_step1, starkinfo_step2, transcript
  - The STARK proof generation and verification code for a given PIL description including encoding of lookup, multiset equality, and connection arguments.

4 Findings

4.1 Protocol Documentation / White Paper Review

4.1.1 Documents Reviewed

- PIL
  - *Polynomial Identity Language (PIL): A Machine Description Language for Verifiable Computation*
  - v1.0, February 13 2023
- eSTARK
  - *eSTARK: Extending the STARK Protocol with Arguments*
  - v1.0, February 10, 2023
  - v1.2, March 15 2023
- Recursion
  - *Recursion, aggregation, and composition of proofs*
  - v1.0 February 13 2023
- fflonk
  - *fflonk: a Fast-Fourier inspired verifier efficient verson of PlonK*
  - ePrint Archive
  - Polygon Documentation (February 20 2023)

4.1.2 PIL + eSTARK

Definition of Connection argument is inconsistent between the PIL and eSTARK v1.0 documents (Informational) The connection argument in Section 1.5 of eSTARK is meant to be identical to Section 1.9, Definition 4 of the PIL document (labeled Multi-Column Copy-Satisfiability). However, Section 1.9, Definition 3 (labeled Connection Argument) refers to the Single-Column variant.

- Suggestion: In the PIL documentation, label Definition 3 to be the Single-Column Connection Argument and specify in the eSTARK documentation in Section 1.5 the description of the Multi-Column Connection Argument.

Definition of Connection argument in eSTARK v1.0 contains inconsistent definition of partition (Inconsistency) The connection argument in Section 1.5 of the eSTARK document defines a partition T={T1,...,Tk}. This implies that T1,...Tk are partition sets. However, there is no need to limit the number of partition sets to k. As described in Section 1.4, Definition 4 of the PIL document, the number of columns can (and usually will) differ from the number of partition sets. We believe this might be the result of a mixup of the partition sets and the vectors that will be used to encode the partition permutation. The number of vectors encoding the partition permutation is the same as the number of columns.

- Concern Addressed: The Polygon team has addressed this concern with version eSTARK v1.2.

Unconstrained selectors in selection variants of lookup and multiset equality arguments (Minor) In Section 2.4 of the eSTARK document v1.0, step 1 states that the prover sends over the selector polynomials. There are two potential issues here:

1. If the verifier does not take in the selector polynomials as constant (preprocessed) polynomials, then the relation is trivially satisfiable. The prover can simply set the selectors to be identically 0. Thus, the Selected Vector Argument is meaningless.
2. Further, beyond the trivial satisfying selection described above, if the prover is meant to send their own selectors, there is no verification check to confirm the image of the selector polynomials over G is {0,1}.

- Suggestion: A self-contained description of the Selected Vector Argument should include a verification check that the selector polynomials lie in {0,1} over G.
- Concern Addressed: The Polygon team informed us that in the zkEVM context, additional constraints exist to ensure the selector polynomials are in {0,1} when they are prover provided, i.e., they are the result of a lookup from a preprocessed table that only contains {0,1}.

Misuse of Poseidon hashing for verifier challenges resulting in unexploitable length-extension weakness (Minor) In Section 2.2 of the eSTARK documentation v1.0, the strategy to produce a third verifier challenge from the 7th, 8th, and 9th field elements requires producing a new set of 8 field elements (the 9th field element is not contained in the previous set). To do this, 8 zeroes are hashed with the previous capacity. This approach produces the same outputs as if hashing the message appended with 8 zeroes: **`M||00000000`**.

- Suggestion: While this particular instance of misuse as used in the STARK prover may not cause issues as the inputs to the transcript may not be length-extended, we recommend that additional verifier challenges are produced by treating the Poseidon hash function as an XOF for continuous variable-length output. Feed the full 12 elements (capacity + output) into another round of Poseidon permutation. Then use the top 8 (without the capacity) as partial output, repeating as needed. This mimics the squeeze operation of a sponge hash function.

Using Poseidon with XOF mode allows for producing as many verifier challenges as needed (Optimization) A number of design decisions are made in eSTARK v1.0 to bound the number of verifier challenges needed. We do not believe there any concerns with the soundness of the proposed design decisions, however if these design decisions were made due to the concern around the feasibilty of generating additional verifier challenges, then this can be avoided. Instead if reducing the number of verifier challenges is already an optimization to reduce the size of the STARK verifier in the recursion, then the following can be ignored.

1. Section 2.2 of the eSTARK documentation v1.0 claims that there is a bound on the number of verifier challenges that can be produced.
2. Section 2.5 of the eSTARK documentation v1.0 claims that the large number of verifier challenges needed to produce the quotient polynomial are hard to produce.
3. Section 2.7 of the eSTARK documentation v1.0 claims that the large number of verifier challenges needed to produce the FRI polynomial is an issue.

- Suggestion: All of the above is not an issue using the XOF mode with Poseidon as described above.
- Concern Addressed: The Polygon team confirmed that the proposed designs to reduce the number of verifier challenges was not because of perceived infeasibility of producing more verifier challenges, rather it is an optimization to minimize the recursive circuit avoiding additional encodings of Poseidon rounds.

Preprocessed polynomials not included in first prover message (Informational) In the description of the full protocol in Section 4.5 Round 1 of the eSTARK paper v1.2, the preprocessed polynomials are not included in the Merkle tree commitment.

- Suggestion: Include the preprocessed polynomials in the first prover message as described in earlier sections.

4.1.3 Recursion

Description of `rootC` used and depicted inconsistently (Inconsistency) In Section 4.2.6 of the recursion document, in Figure 17, the public input **`rootC`** is supposed to be multiplexed with a hard-coded **`rec1 rootC`**. The paragraph before has some description mistakes.

- Suggestion: Follow the diagram from Jordi's slides that includes the hard-coded **`rootC`** and input **`rootC`** as argument to the multiplexor fed to both verifiers.

4.1.4 fflonk

Minor soundness bound error (Informational) There is an off-by-one error int he soundness bound for the proof of Lemma 6.4. Note that $A \mid (p-1)$ where p is the size of F. The set S of A-th powers of F includes 0. Thus, $|S| = \frac{p-1}{A} + 1$. Let F_i be a non-zero polynomial. For a uniformly chosen element of s in S, the probability

$$f_i(s) = 0 \leq \frac{\text{def } F_i}{\frac{p-1}{A} + 1} < \frac{\text{def}(F_i) \cdot A}{p-1}$$

4.2 Code Review

4.2.1 Scope of Code Review

Code Reviewed

1. Recursion pipeline
   - Repository: zkevm-proverjs at branch **`v0.8.0.0-rc.2-forkid.2`**
   - Build documentation
   - The generated STARK verifiers in the recursion pipeline for c12a, rec1, rec2, recf, and fflonk.
2. STARK prover and verifier for PIL
   - Repository: pil-stark
   - Files: stark_gen, stark_setup, stark_verify, starkinfo, starkinfo_cp_prover, starkinfo_cp_verifier, starkinfo_fri_prover, starkinfo_fri_verifier, starkinfo_step1, starkinfo_step2, transcript
   - The STARK proof generation and verification code for a given PIL description including encoding of lookup, multiset equality, and connection arguments.

Future Code Reviews While the scope of the broader zkEVM security review included a complete code review of all cryptography related parts of the zkEVM, the code review of the following parts have not been done yet (at the time this report was written) but will be completed in the near future.

1. Compilation of PIL file to data structure input to STARK prover
   - Repository: pilcom
2. Encoding of PIL-derived STARK verifier to circom
3. Encoding of circom-produced R1CS to PlonK-like PIL
4. FRI (batched) polynomial commitment and opening

4.2.2 Recursion Pipeline

Following the latest build documentation provided, we built **`zkevm-proverjs`** on branch **`v0.8.0.0-rc.2-forkid.2`**. We checked that all the circom files aligned with out analysis of the recursion pipeline as described in the recursion documentation. Additionally, we reviewed the final fflonk verifier in Solidity.

Build and documentation mismatch (Minor) The **`final.fflonk.verifier.sol`** produced by zkevm-proverjs on branch **`v0.8.0.0-rc.2-forkid.2`** uses an outdated template from snarkjs before this commit. This outdated template does not appropriately calculate verifier challenges (does not include preprocessed polynomials and does not keep a continuous transcript state). Further the documentation provided in the README is outdated.

- Suggestion: The Polygon team should double check the release version of zevm-proverjs uses the updated template otherwise the produce fflonk verifier may have Fiat-Shamir vulnerabilities.
- Suggestion: Use the updated documentation provided by Felicia.
- Concern Addressed: The Polygon team confirmed that they are using the correct version in their release build pipeline.

Batch number comparison logic overflow (Minor) In the recursive2 circuit, a mux is used to select whether a hardcoded root or public input root is passed into the subverifier circuits. In the recursivef circuit, a mux is used to select which of two hardcoded roots is passed to the subverifier. Both muxes take in as input a comparison of the form:

```
component test = IsZero();
test.in <== oldBatchNum - newBatchNum -1;
....
mux.s <== test.out;
```

Since the bn128 scalar field is larger than the goldilocks field, there is a potential that overflows occur in the field arithmetic above. This would affect the comparison logic above done over the goldilocks field.

- Example: Let p be the order of the larger bn128 scalar field, and let q be the order of the goldilocks field. Note $p > q$ as the **`bn128`** field is much larger than the goldilocks field. Define integers a and b and consider expressions $a \cdot q + b$ and b. Then, $aq + b \equiv b \pmod{q}$ yet $a \cdot q + b \neq b$ as integers. If **`oldBatchNum`** is $b - 1$ and **`newBatchNum`** is $a \cdot q + b$, then the comparison above would return true and despite the batch numbers not being adjacent. The prover could convince a verifier that they have done $a \cdot q + 1$ iterations of work by using a valid final proof of an execution from $b - 1$ to b.
- Concern Addressed: The Polygon team stated that in **`final.verifier.circom`** they do a bit composition of **`oldBatchNum`** and **`newBatchNum`** into 63 bits. Since $2^{63} < q$ (the order of the goldilocks field), and overflow in the comparison logic cannot occur.

Minor completeness issue by rejecting the point-at-infinity during on-curve checks (Informational) Valid points on the curve bn128 either satisfy the curve equation or are the point at infinity. In the precompiled contracts (EIP196), (0,0) is the encoding of the point-at-infinity. The check checkPointBelongsToBN128Curve function returns true only if the point satisfies the curve equation; it rejects the point-at-infinity. There exists valid scenarios where a prover message is identical to the point-at-infinity.

- Suggestion: Add the point-at-infinity check to the on-curve check.
- Concern Addressed: The Polygon team would like to avoid adding the on-curve check for the point-at-infinity. We have confirmed that due to the random blinding factors and random verifier challenges in the protocol, completeness would only be an issue with negligible probability for an honest prover.

4.2.3 STARK Prover and Verifier

We checked the implementation of the STARK prover and verifier and generation of the **`starkinfo`** object from the parsed PIL object. We verify that the STARK prover follows the design of the reviewed eSTARK protocol, in particular, the implementation of the newly added lookup, multiset equality, and connection arguments.

Misuse of Poseidon hashing for verifier challenges (Minor) We repeat the issue raised during the cryptography review above. It is represented in transcript (lines 16-20). The suggestion and addressed concern are the same.

Unnecessary inclusion of all committed polynomials in FRI polynomial (Optimization) When the prover computes the FRI polynomial in starkinfo_fri_prover (lines 12-19), all committed polynomials are included in the linear combination. All committed polynomials that are involved in constraints will already be included in the linear combination as part of the proof for their correct evaluation, which will attest to the degree bound for the committed polynomial. Thus, the only benefit of adding all committed polynomials to the linear combination is to include a degree bound check for polynomials that are not part of any constraints.

- Suggestion: Committed polynomials can be removed from the linear combination computing the FRI polynomial. If for some reason it is important to check the degree bound of some committed polynomial that is not included in any constraints, then at the very least, all committed polynomials that are a part of constraints do not need to be included a second time.

Unnecessary evaluation of vanishing polynomial in verifier (Optimization) The verifier evaluates the vanishing polynomial of G on $g \cdot z$ where g is the generator of G and z is the verifier evaluation challenge. However, this evaluation is not needed for verification. The evaluation is in stark_verify (line 70).

- Suggestion: Remove this evaluation.