We are from the team at EZKL, and developers of the open EZKL library which generates zero knowledge proofs for algorithms, data science models, and machine learning models.

In the past few weeks, we've been working to prove correctness of the Retroactive Public Goods Funding results. We've piloted the proof with Round 3. This post intends to explain the mechanics of the audit and also ask for feedback from the Optimism community.

The TLDR;

- The Retroactive Funding process and algorithm ideally improves with each round via governance feedback.
- We've essentially audited RetroPGF Round 3. We did so through a cryptographic proof, and anyone can verify it using this webapp.
- This paves the way for an end-to-end automated process with private voting, checks and balances on the allocation algorithm, and smart contract enabled allocation delivery.

For stakeholders, this audit means many things:

- For badgeholders, that your vote was fairly counted
- For recipients, that your allocation was fairly calculated
- For everyone else, the results actually correspond with the Optimism governance forums

An iterative approach

As this forum is aware, a core tenet of the Optimism community is agile iteration. That means the design of Round 3 looks much different from that of Round 1. We see that Round 3 heavily indexes on the impact evaluation framework, and as a both qualitative and quantitative measure, heavily relies on the judgement of voters (badgeholders).

Votes are currently fed through an allocation algorithm, published publicly by the Optimism team. The team announces the algorithm's results at the conclusion of a round, as well as learnings to iterate upon for the next rounds.

The trickiest bit has always been effectively measuring impact. Reliance on quantitative metrics risks exclusion. Reliance of qualitative feedback risks human biases among other coordination challenges. And there are many more tradeoffs beyond these. Consequently, the algorithm and overarching design continues to evolve and evoke discussion. Ideally, we can check the process and algorithm with each new iteration.

The integrity challenge

One of the main tradeoffs for Retroactive Funding is related to privacy.

For such a large pool of OP rewards, transparency is important. We want to ensure results are aligned with previous governance decisions. More importantly, this recognizes that many projects depend on the allocation for sustainability.

However, badgeholder votes are private in Retroactive Funding. A longstanding feature of voting systems, this allows voters to express their true preferences without fear of retaliation, coercion, or pressure from others. It makes it difficult to execute vote buying, and encourages honest participation.

This presents a key challenge

: verifying that allocation outcomes are truly based on badgeholder votes. Even though the allocation algorithm is public (and we can check that it accurately codifies forum promises), we can't run a check without the input data. In the worst case, we can't guarantee that the provided allocation algorithm was even used or that all votes were counted.

The zero knowledge solution

Making this process transparent requires both a proof that legitimate votes were counted and that the allocation algorithm was run as intended, whilst keeping the votes themselves private.

This is exactly the sort of proof zero-knowledge (ZK) cryptography was designed to enable — and why we're so excited to showcase this project as a hallmark use of the technology.

There are two places we apply ZKP:

1. Verifying the voting process

The first circuit verifies that the cast ballots are legitimate, and that all legitimate ballots are counted. Ballots are legitimate if signed by a badgeholder from the public whitelist.

The details:

1. Check the ballot signatures against the whitelist.
2. For each project, create a hash of all its legitimate ballots. This is denoted as Hash A.
3. Aggregate the hashes to generate an overall proof. This proof means that across all projects, all legitimate votes were counted.

This results in Proof A.

2. Checking the allocation algorithm

The second circuit verifies the usage of the allocation algorithm.

The details:

1. For each project, we repeat steps 1 and 2 of the first circuit. This, however, is denoted by Hash B. We will check that Hash A = Hash B out of circuit.
2. Generate a proof over the allocation algorithm. The algorithm takes in a ballot table as input, and outputs an allocation table. We only need one proof of the whole algorithm.

The results in Proof B.

3. Combining the proofs

Since we generate the proofs separately, we want to link them together cryptographically. This ensures there is not tampering of votes in between proofs. We link them by checking that the hashes of legitimate votes, Hash A and Hash B, are the same. This is the same hash you see on the platform, just in QR form.

4. Verifying a project's allocation

This is where you come in! When you press verify

, you check the allocation was correct for a given project.

Even though the proofs are for all projects,

each proof has a set of public variables particular to a given project. In Proof A this is the hash of the votes (Hash A), in Proof B this is the hash of the votes (Hash B) and

the allocation amount for a given project.

If we change any of those variables, we break verification for a particular project

. Try it yourself!

5. Verifying the entire RetroPGF

It's also possible to verify the entire RetroPGF round at once. You can do so by pressing "verify" on the landing page.

Everyone's verification matters

If you're neither a badgeholder nor a recipient, why hit the verify button? Simply put, verifying the results of the round contributes to the security and legitimacy of the process.

Here's the idea

: Blockchains have both social consensus (which fork is the main one, which client should be run, etc.) and computational consensus. A smart contract can enforce a set of rules, but humans decide which copy of the smart contract is the canonical one. Election workers count ballots, but citizens recognize the result. Similarly, by verifying the RetroPGF results for one or more projects, you help solidify the social consensus around the legitimacy of the vote. You are an election observer, contributing to a successful audit.

Implementation details

You can find the code for this on ourblog. We briefly discuss the details for verification of the allocation algorithm.

Optimism's allocation algorithm was written in Python, which made the EZKL python library a natural fit for in-program

conversion of the data science into zero-knowledge proofs. You'll also notice that OP allocation votes are cast with floating point amounts — this meant a very low tolerance for imprecision.

EZKL already optimizes heavily for precision given the frequency of nonlinear operations (reciprocals, softmax, etc.) in data science and machine learning models. However, we also made substantial improvements to the core engine, and specifically how we perform lookups, in order to retain proof size while increasing numerical accuracy. Incidentally, proving performance for other models like random forests and transformers improved significantly (sometimes over 90%). A win-win!

There are a few more important details, specifically around signature verification and the proving system. This may require some background in cryptography, so we omit a full discussion here. However, this also is available on our blog.

Next steps

This is just the first step in auditing and automating the Retroactive Funding process. In the future, it is possible to create more comprehensive systems and even automate the process end-to-end. This may look something like:

- Algorithm implemented and formally verified against specifications in governance forums
- Votes checked against whitelist of badgeholders via a proof of legitimacy
- Upon verification, votes run through the allocation algorithm
- Final allocation checked via a proof of calculation
- Upon final verification, distribution of funds automatically

This also makes it easier to reflect and analyze voter behavior without revealing information about any one badgeholder — facilitating incremental improvement for the next round.

Open questions & call for feedback

We would be grateful for any feedback on the platform, and will aim to implement improvements for future rounds. In particular:

- Were you more interested in the verification of the whole RetroPGF or focus on particular projects?
- Did you find features like the ability to manipulate the hash or allocation amount interesting? Did they make the meaning of the proof more clear?
- Do you feel comfortable with the implementation, or should we provide further details?
- Are there additional aspects of the voting and allocation process where you would like to see this applied? For example:
- Verifying data analysis on votes/results
- Verifying algorithms to select badgeholders or legitimate voters
- Verifying the weight / inclusion of certain impact metrics
- And any other ideas you may have!
- Verifying data analysis on votes/results
- Verifying algorithms to select badgeholders or legitimate voters
- Verifying the weight / inclusion of certain impact metrics
- And any other ideas you may have!

We look forward to feedback and further discussion!