Proposal: On Comparing Proof Systems and their Implementations

(or "It would be cool to have the L2BEAT of proof systems!")

Matteo Campanelli and Marco Stronati

Matter Labs
{matteo,ms}@matterlabs.dev

Abstract. The goal of this proposal is to renew interest in the comparison of proof systems, as already extensively discussed by Benarroch et al. (3rd ZKProof Workshop) [BNTT20] in 2020, whose proposal however saw little adoption in the community.
We propose a framework for the submission of proof-systems implementations, where each submission consists of a software artifact implementing a small set of high-level circuits (such as SHA256) and a specification document describing the architecture of the proof system and its security analysis (to make sure all systems achieve the same target).
Each submission artifact is benchmarked on a single standard machine and each specification is peer-reviewed by members of the community.
All submissions have their results displayed on a website which allows for comparison of quantitative measure (such as proof size) as well as qualitative properties (such as relying on well established primitives).

1 Scope and Outline

The aim of this document is to stimulate a discussion around comparison of ZKPs. [1] While we put forward a concrete proposal that we believe could bring some progress on the topic, we acknowledge its limits, welcome feedback and expect it to evolve significantly over time. *Our goal is to make explicit what we see as obvious gaps in what is available to researchers and practitioners in the community.* We do not claim ours to be an exhaustive discussion about benchmarking practices.

Benarroch et al. [BNTT20] discussed approaches to comparing ZKPs. In this document we first briefly expand some of the points in that document. This first part focuses on benchmarks for performance. Next we make a proposal for a review-based proposal for comparing ZKP systems with implementations. In this second part we are more explicitly interested in other properties of systems as well, e.g., security of the assumptions and quality of artifacts.

[1] We shall write "ZKPs" from now whenever we refer to cryptographic proof systems in general even if we do not strictly require *zero-knowledge* in the technical sense.

2 Motivation

Running fair comparisons among different schemes *easily* is important to the ZK community:

- for *researchers*, who desire to compare their new designs to existing ones or would like to critically assess published work or work submitted to publication.
- for *practitioners*, who may be seeking for existing systems to adopt.
- for *consumers* of applications of ZKPs, who want to compare providers. This is especially urgent in the rapidly expanding space of Layer 2 using validity proofs (e.g. ZK-Rollups).

Being able to compare systems drives the necessity to document them and the development of common interfaces, abstractions and standards. Since innovation in the space often means *better performance* benchmarking seems a natural comparison ground. However these systems are increasingly complex and subtle. A transparent specification of how they work and especially of their security is even more relevant today than their performance profile. Furthermore one can easily beat a benchmark by degrading security, so benchmarks and specification are two necessary components of any meaningful comparison.

3 Recap of [BNTT20] (3rd ZKProof Workshop), minor extensions, current state of affairs

The following were challenges discussed in [BNTT20] around benchmarking of ZKPs:

1. Identifying functionalities for the benchmark.
2. Allowing solution designers to demonstrate progress on both front ends and back ends (and their combination), while at the same time not artificially tying the hands of protocol designers to insist that they use intermediate representations or work in the "frontend + backend" paradigm.
3. Questions surrounding specificity in both the chosen functionalities and the solutions for those functionalities.
4. Allowing/encouraging researchers to demonstrate complicated tradeoffs between costs, to demonstrate concrete performance on fixed statement sizes, and to demonstrate scalability.
5. Proper cost accounting.
6. Comparing solutions in a manner that is not overly dependent on the operation environment.

One of the solutions proposed in [BNTT20] was to adopt *driver* and *analysis* tools to test a system and measure its performance. The document also proposed the standard proposal zkInterface [BGK+20] to instantiate these tools.

3.1 Comparison with this proposal

We agree with the core of the discussion in [BNTT20] and it forms the foundation of this proposal, we feel the need on one hand to update it and on the other to position ourselves with respect to it.

- Since 2020 the community saw significantly more proof schemes and arithmetization approaches. Plonkish [hBc] in particular saw wide adoption and, due to its flexibility, has proven hard to standardize. For this reason we consider high-level black-box comparison as described in section 4 as a good starting point.
- Efforts like zkInterface should be extended to new arithmetizations formats besides R1CS and we envision a future iteration of our proposal that evaluates proof-system based on a common set of circuits. Unfortunately this is further away in the future.
- While [BNTT20] describes in depth what should be evaluated in the comparison of proof systems, we propose a human process to achieve it.

3.2 Current state of affairs for comparing ZKP systems

The last years have seen a flourishing not only of new constructions for concretely efficient ZKPs in academic conferences, but also of the deployment of variants of such systems.As a consequence we see an even larger divide between practices within the community.

- **"Academic" ZKP constructions:**
  (pro) tend to have explicit security statements, proofs, constructions.
  (pro) are peer-reviewed.
  (con) may lack an implementation or one that is usable in practice without substantial changes[2].
- **Deployed ("Industry") constructions:**
  (pro) may be used in practice.
  (pro) tend to be highly optimized.
  (con) tend to be described only informally and in non peer-reviewed white papers.
  (con) rely on security audits which have debatable effectiveness (there are very few experts with sufficient knowledge of ZKP schemes).
- **In neither of the above cases** it is easy to compare systems "at a glance". A reliable comparison may not be available or it may be not readily available to consumers (e.g., maybe the comparison is buried in a wordy *Related Work* section).

While the divide above is unsurprising, we strongly propose that it should be bridged: there should be a system that combines (and improves upon) the best of both worlds to assess ZKPs.

[2] We are not arguing that all academic publications should include a "real-world" implementation. This point intends to stress the divide between constructions that are proven secure and those that are actually implemented and deployed.

4 Proposal for a peer-review based comparison of ZKPs

We propose a process based on submissions consisting of a software artifact and a specification document. Both the artifact and the document should be assessed and reviewed by experts in the community. The process could be organized in 4 rounds of submission throughout the year, similarly to what is done in some academic conferences. This could allow to organize a review process with fast turnaround time and with a light load for reviewers. It is crucial for the result of the process to be published in an easily accessible format, such as a website to interactively compare proof systems, in order to inform users and incentivize new submissions (see also "Interface" below).

The scarcest resource for our proposal is expert reviewer time. For this reason we propose to set the expectation on the quality of submission based on available review time. For example if we ask reviewers for a commitment of a week per round fully dedicated to reviewing submissions and we anticipate 1-2 submissions per reviewer, then this sets the amount of work that we can expect in each submission. There is no point in having a highly detailed specification of 20 pages if nobody is going to read it.

This should not discourage us from accepting partial or lower quality submissions, so long as this can be clearly displayed in the resulting review. For example a submission's artifact that does not implement all required payloads or that is missing the specification of one component could still be accepted.

We should also encourage the already reviewed proof system to update their submissions at each round. Given more review time a submission could increase in complexity and comprehensiveness. For example the first submission of a proof system could only contain one benchmark payload and a high-level description of the protocol. After 4 rounds of reviews we would expect a submission to cover all payloads and provide a detailed specification of each component of the protocol.

The competition for reviewer time should also encourage code and specification reuse. For example, if a protocol builds on an polynomial commitment scheme that has already been reviewed in the past, then there is much less time needed to review this component, time that can be spent elsewhere. Hopefully over time we will see the emergence of software libraries that went through many rounds of review and are re-used by many proof systems.

Artifact evaluation One of the goals of the review process should be to assess performance features of the submissions. We refer the reader to [BNTT20] and [ECK+23] for general discussions around comparing ZKP systems.

Benchmarking should be easily reproducible so we propose to limit the scope to a single machine that can be rented on popular cloud platforms. The submission should include instructions for artifact assessment as done already in certain conferences. This design decision is consciously trying to be less ambitious than necessarily requiring full compatibility with a driver tool (as in [BNTT20]), which could be optional but recommended for submissions. If certain models of GPUs are widely available then they can be used.

We purposefully exclude systems that require a cluster of machines to parallelize proof generation. These systems are too intricate to review in sufficient detail and too complex to reproduce experimentally. Systems relying more heavily on hardware optimizations, such as specific GPU, FPGA, ASICs etc are also out of scope for this proposal for complexity reasons and also to avoid overlap with existing initiatives like ZPrize [zpr].

At least for the first rounds of submission we encourage a block-box style of benchmarking, by which we mean that, given a set of payloads such as SHA256 over a specific input size, a submission can prove and verify the required payload using any form of circuit representation or proof system configuration.

As an example submission, one could use Halo2+KZG to implement directly SHA256 in circuit, while another submission could use a zkVM to implement SHA256 in Rust and then use a STARK prover followed by a Groth16 prover to reduce proof size. Clearly the second solution trades off prover time for proof space, if that is the metric the designers are more interested in. However a more complex system incurs also a higher specification complexity, the latter submission will need to specify two proof systems and argue the correctness of the STARK verifier written in circuit.

In the future we could introduce more categories as we see fit. For example to stimulate the development of client-side proving, we could have a category that requires compilation to WebAssembly, a limit of 2GB of memory and 2 CPU cores.

Review of specifications We require submissions to provide a formal version of the specifications of their system. An ideal example in this sense is the ZCash protocol specifications [HBHW]. At the same time we imagine less detailed submissions to be acceptable for review. Specifications should include details about the ZKP system used, whether it relies on other common building blocks, how it is currently instantiating cryptographic primitives, and security claims. We expect reviewers to check at least superficially to which extent the specifications match the artifact.

Reviewers should also verify security claims. Here the main goal of peer reviews would be to establish some useful graph dependency for the security assumptions in that system. This already occurs to some extent in academic papers.

What we are proposing here, however, differs in two ways. First, we see it important to express this chain of dependencies as formally as possible. In other words, there should be an explicit *ontology* of "assumptions" so that one could formally express (and visualize) connections among them. Second, the security of specific constructions is more nuanced than generic cryptographic protocols. Therefore what we mean by "assumptions" is quite general. The dependencies we have in mind encompass for example:

1. general statements of security (e.g. "System X is secure under the DLOG assumption")

2. reliance on other systems in the same review framework (e.g., "ZKP system X relies on the security of ZKP system Y").
3. more idiosyncratic security points (see below)

The second item can be useful since sometimes upgrades in one system replace specific building blocks with some other[3].

The last item include examples such as:

- "this specific curve offers *foo* bits of security against Pollard-Rho attacks".
- "this specific hash function is a random-oracle"[4]
- "each verifier query in system *foo* adds *bar* bits of security" (see [Tha] for an example of this type of assumptions in prominent systems)

What is the profile of whom we envision participating in reviews? Verifying security claims will require the knowledge of cryptographic experts from academia and elsewhere. Their profile could be that of reviewers on ZKP topics for conferences such as USENIX Security, IEEE S&P, ACM CCS, IACR CHES. Artifact assessment should probably involve security engineering experts in the ZKP community. See also discussion below in "Incentives".

Interface The interface should allow to explore the results of each round of submission, showing qualitative and quantitative measures of each system. It is important to encourage a rich comparison of ZKPs across all measures, without incurring in simplifications like which one has the shortest prover time. The goal is to improve transparency and not encourage submissions to optimize for a single measure in order to climb a leaderboard.

A useful analogy in the L2 blockchain world is the website `https://l2beat.com/` which compares widely different systems across many values[5]. One aspect which we feel should be improved upon is reducing the leaderboard style strict ordering of submissions. A particularly pathological example is the "Risk Analysis" section where incomparable properties related to security are still presented as a total order.

There are other features that it would be important to display besides the sheer output of reviews. For examples whether a submission has not been updated for a long time, if it lacks components or if it has gone through multiple reviews.

Incentives As already stated, reviewer time is scarce and it is hard to ask developers under pressure to put a system in production to invest time in preparing

[3] One such example is the difference between Halo and Halo2. See here: `https://electriccoin.co/wp-content/uploads/2020/09/Halo-puzzle-03-scaled.jpg`

[4] This is a common example, but we also acknowledge that random oracles are controversial objects. See [CGH98] and [Gre].

[5] Other examples, respectively in the world of programming languages and antivirus software are [Con] and [Ins]

a submission. A hope for success of this system would be some type of *snow-ball effect*: If the website—the one presenting the result of these reviews publicly—were to become an important point of reference for the community, then having a good looking submission becomes valuable for companies and academics alike. At the same time, if a large number of systems were to be represented, then being absent would be a red-flag.

As for reviewing, it could become a chance to be in contact with the best academics and practitioners in the ecosystem and being up to date with the most recent development in the space. There could also be the (soft) requirement that for any submission received, authors would be expected to put in some review time of their own.

It should be noted that, while the majority of submissions would be prepared by the main developers of a system, this need not be the case. Anybody can prepare a submission for a proof system, even if the original authors may know better how to optimize or specify it. This would be a way to signal to the authors that other people see as valuable that their systems are assessed within this framework.

5 Conclusions

Were the community to accept this proposal positively, there would be *many* challenges in realizing it effectively. Some of them have to do with finding resources to bring about a Proof of Concept on it and to keep improving on it consistently. So far the ZKProof community has had mixed results with the latter on proposals of this type. Some other challenges are specific to the vision we sketched. One of the biggest risks we envision is that the perceived value of the system is made invalid by too many submissions without further reviews. Another is how to bootstrap this process in order to obtain the snow-ball effect mentioned earlier.

This document is not a declaration of intentions. Our main goal is to push for *some* proposal that solves the challenges outlined here and in [BNTT20]. We *welcome* further comments on the vision sketched here. If there were to be some form of convergence, we would be open to work on it *together* with the rest of the community.

Acknowledgments The authors thank Denis Firsov, Benjamin Livshits, Kobi Gurkan, Stefanos Chaliasos, Mary Maller and Jonathan Rouach for useful discussions.

References

BGK+20. Daniel Benarroch, Kobi Gurkan, Ron Kahat, Aurélien Nicolas, and Eran Tromer. Community proposal: Zkinterface, a standard tool for zero-knowledge interoperability. *QEDIT, Tel Aviv-Yafo, Israel, Tech. Rep*, 2020.

BNTT20. Daniel Benarroch, Aurélien Nicolas, Justin Thaler, and Eran Tromer. Community proposal: A benchmarking framework for (zero-knowledge) proof systems. *QEDIT, Tel Aviv-Yafo, Israel, Tech. Rep*, 2020. URL: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-benchmarking.pdf.

CGH98. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. Cryptology ePrint Archive, Paper 1998/011, 1998. https://eprint.iacr.org/1998/011. URL: https://eprint.iacr.org/1998/011.

Con. Benchmarks Game Contributors. Which programming language is fastest? https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html.

ECK+23. Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. *Cryptology ePrint Archive*, 2023.

Gre. Matthew Green. What is the random oracle model and why should you care, part 5. https://blog.cryptographyengineering.com/2020/01/05/what-is-the-random-oracle-model-and-why-should-you-care-part-5/.

hBc. The halo2 Book contributors. Plonkish arithmetization. https://zcash.github.io/halo2/concepts/arithmetization.html.

HBHW. Daira Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specifications. https://zips.z.cash/protocol/protocol.pdf.

Ins. The Independent IT-Security Institute. AVTest. https://www.av-test.org/en/antivirus/home-windows/.

Tha. Justin Thaler. SNARK security and performance. an aside: Diving even deeper into pq-snark security. https://a16zcrypto.com/posts/article/snark-security-and-performance/#section--9.

zpr. ZPrize Competition. https://www.zprize.io/.