

# Servicifying zk-SNARKs Execution for Verifiable Off-chain Computations

Alvaro Alonso Domenech, Jonathan Heiss, Stefan Tai

Information System Engineering

TU Berlin

Berlin, Germany

{aa,jh,st}@ise.tu-berlin.de

**Abstract**—Zk-SNARKs help scale blockchains with Verifiable Off-chain Computations (VOC). zk-SNARK DSL toolkits are key when designing arithmetic circuits but fall short of automating the subsequent proof-generation step in an automated manner. We emphasize the need for portability, interoperability, and manageability in VOC-based solutions and introduce a Proving Service that is designed to provide a scalable and reusable solution for generating zk-SNARK proofs leveraging clouds.

**Index Terms**—Blockchain, Zero-knowledge Proofs, zk-SNARKs, ZoKrates, Proving, Service Architecture, Cloud Computing

## I. INTRODUCTION

The succinctness property and the short verification times of zk-SNARKs come at the cost of large computational complexity, static execution models, and memory overheads during the proof generation process [1]. These limitations represent a problem in VOC applications which require handling large and varying workloads, like rollups. This type of application would benefit from scalable, interoperable, and manageable system environments like clouds. However, DSL toolkits like ZoKrates [2] or Circom [3] concentrate on circuit development and put little emphasis on how to integrate these circuits into production systems. Bridging the gap between advanced cryptography and modern systems engineering also helps to create standardized benchmarks for proving systems and tools.

## II. PROVING SERVICE

We introduce a service-oriented approach for VOC that facilitates the use of the cryptographic procedures of zk-SNARKs within cloud system architectures. Our system allows for executing arithmetic circuits as encapsulated application logic in containers that are deployable to different machines realizing scalability, provide interoperability with other services, and are better manageable. For that, we give an overview of the service-oriented system architecture and describe the internals of the proving service.

### A. Service Oriented Verifiable Off-chain Computation

The problems above can be fulfilled through a service-oriented approach as depicted in Figure 1. Starting from a higher-level system’s perspective, we treat the proving service as a black box which, upon a request, returns the proof together

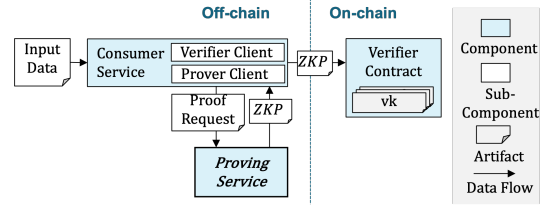


Fig. 1: Proving Service Model

with the computation’s output. Following the VOC model [2], we distinguish between the blockchain infrastructure hosting the *verifier contract* and a cloud-based off-chain infrastructure that runs outside the consensus protocol and hosts the *consumer* and the *proving service*.

The *consumer service* is responsible for managing the proving service’s inputs and outputs and interacting with the relying verifier contract. It receives data from external sources and translates them into a *proof request*. Upon a request through the *prover client*, the proving service executes the VOC and returns the *ZKP* attesting to the computational integrity of the VOC. The consumer service then submits the *ZKP* to the verifier contract through its *verifier client*. On submission, the verifier contract verifies the proof computed by the proving service.

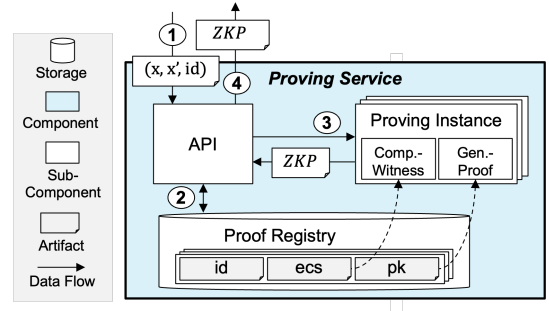


Fig. 2: Proving Service Architecture

### B. Service Architecture

The *Proving Service* depicted in Figure 2 is an application service that runs stand-alone on the prover’s off-chain infras-

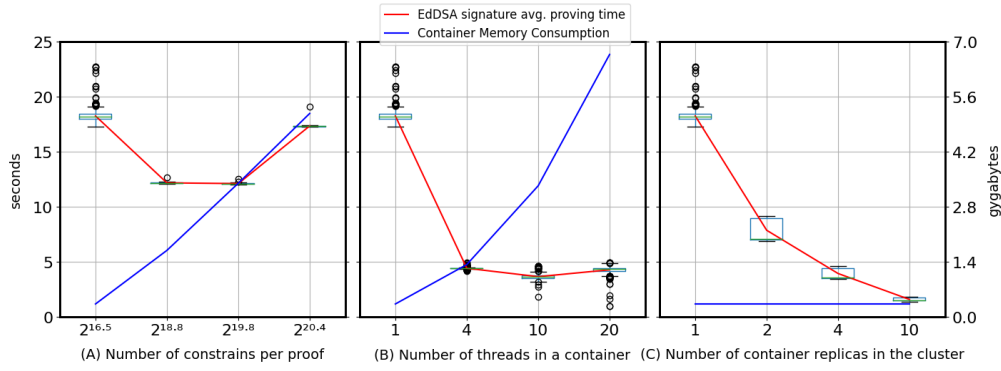


Fig. 3: Proving Time and Memory Consumption

structure. It serves proof requests by exposing the proving-related operations through an *Application Program Interface* (API) as minimal functionality to the consumer service.

The procedure can be summarized in four simple steps: First, the proving service receives the *Proof Request* containing the public ( $x$ ) and private ( $x'$ ) proof arguments and the identifier of the addressed Executable Constraint System (*ecs*) (*id*). Second, using the *id* the API fetches the corresponding *ecs* and *private key* (*pk*) from the *Proof Registry* which is the persistent storage component containing these large, recurrently requested files needed for proving. The proof registry manages different reusable pairs of *ecs* and *pk*, each addressable through a unique *id*.

Third, the *Proving Instance* is executed in two stages: the witness computations and the proof generation. Fourth, the ZKP is returned to the consumer service through the API upon successful execution.

### III. ZOKRATES-API

The previous service-oriented architecture serves as a technology-agnostic blueprint for building proving services for DSL circuits. For evaluation, we technically instantiate the proving service for ZoKrates [2] and present the ZoKrates-API<sup>1</sup> as a ready-to-use open-source software. We servicify ZoKrates by wrapping an API around the ZoKrates interpreter, the central component of the ZoKrates software that previously has only been addressable through a Command Line Interface and a Javascript library. The ZoKrates-API exposes the methods of the ZoKrates interpreter through HTTP endpoints. We containerized the ZoKrates-API using Docker, making the services easily deployable among a wide range of machines and allowing us to further leverage cloud-native tools like Kubernetes for horizontal scalability, manageability, and observability. Furthermore, the ZoKrates-API supports multi-threading so a single instance can compute multiple proofs in parallel.

### IV. EVALUATION

To test the presented implementation, we deployed the containerized ZoKrates-API on a Kubernetes cluster. As a

workload for our experiments, we generated a large number of EdDSA signatures [4] which amounts for more than  $2^{29}$  of circuit constraints similar to [5].

We conducted three experiments (see Figure 3) to measure the average proving time per signature in [sec] and the memory consumption in [gb] using various cluster configurations. Figure 3A) shows a 33% improvement in proving time for a single machine when choosing an appropriate machine size. An argument in favour of vertical scalability. For a single machine as well, Figure 3B) shows that the proving time can be brought down drastically when few parallel threads are enabled, though the gains plateaued rapidly due to the increasing resources needed. Running the same experiments in parallel VMs instead of threads, Figure 3C) demonstrate a better approach to scaling proving as the computational burden is distributed over several machines, proportional increasing the processing time.

### V. CONCLUSION

As the experiments demonstrate, significant performance improvements can be gained from horizontal (more nodes) and vertical (larger nodes) scalability of an arbitrary zk-SNARK proof. We facilitate this process by leveraging modern cloud-native architectures such as Docker and Kubernetes.

### REFERENCES

- [1] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, “Scalable zero knowledge via cycles of elliptic curves,” *Algorithmica*, vol. 79, pp. 1102–1160, 2017.
- [2] J. Eberhardt and S. Tai, “Zokrates - scalable privacy-preserving off-chain computations,” in *IEEE International Conference on Blockchain*. IEEE, 2018.
- [3] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, “Circom: A circuit description language for building zero-knowledge applications,” *IEEE Transactions on Dependable and Secure Computing*, 2022.
- [4] J. Heiss, A. Busse, and S. Tai, “Trustworthy pre-processing of sensor data in data on-chaining workflows for blockchain-based iot applications,” in *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings 19*. Springer, 2021, pp. 133–149.
- [5] A. Chiesa, R. Lehmkuhl, P. Mishra, and Y. Zhang, “Eos: Efficient private delegation of zkSNARK provers,” in *USENIX Security Symposium*. USENIX Association, 2023.

<sup>1</sup><https://github.com/ZK-Plus/zokrates-api>