Servicifying zk-SNARKs Execution for Verifiable Off-chain Computations

***Abstract*—Zk-SNARKs and their use for Verifiable Off-chain Computations (VOC) have proven effective in enhancing the privacy and scalability of blockchains. However, their computational overheads and static execution models clash with application-driven requirements for efficient handling of large and varying workloads. Current tooling does not provide adequate support for developers to solve this conflict. Addressing this gap, we propose a service model for the execution of zk-SNARKs, thereby integrating VOCs with modern serive-oriented system engineering practices and improving their scalability, interoperability, and manageability. As a technical evaluation, we present the *ZoKrates API* as a ready-to-use open-source proving service and evaluate it through initial experiments on typical VOC workloads.**

***Index Terms*—Blockchain, ZKP, zk-SNARKs, ZoKrates, Proving, Servicification, Cloud Computing**

I. Introduction

In recent years, zk-SNARKs have consolidated as a central technology to solve privacy and scalability concerns of blockchains [1] due to their proof conciseness and cheap verification [2]. When used for Verifiable Off-chain Computation (VOC) [3], they enable computations outside the blockchain's consensus protocol without compromising computational integrity. As such, zk-SNARKs have been applied in various approaches to oracles [4], relays [5], or rollups [6].

However, 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 [7]. These limitations represent a problem in VOC applications which require handling large and varying workloads, often on heterogeneous machines. Examples are rollups that become more economical the more transactions are processed at once but need to adapt to the load of incoming transactions to reduce idle times and guarantee liveness.

Application-driven requirements demand suitable computational models and technological support to integrate zk-SNARKs in modern application systems and engineering practice, and to let them benefit from scalable, interoperable, and manageable system environments like clouds. However, there are no guidelines or models on how to bring zk-SNARKs-based solutions to such production-friendly environments.

Toolkits like ZoKrates [3], Circom [8] or Noir [9] provide valuable support for developers to create zk-SNARKs-based applications. But to date, they rather imitate cryptographic processes and lack support for cloud-driven development and management. Bridging the gap between advanced cryptography and modern systems engineering would also help to create standardized benchmarks for proving systems and tools as recent efforts demonstrate [10]–[12].

Addressing this gap, in this paper, we provide the following individual contributions:

1) We provide an analysis of the intrinsic limitations of zk-SNARKs in the context of VOC applications. We identify a conflict between application requirements and zkSNARKs limitations that is not adequately addressable through available tooling.
2) We introduce the concept of a *Proving Service* that allows for exploiting scalable and manageable cloud infrastructures for zk-SNARKs executions facilitating their integration into modern application systems.
3) We present the ZoKrates API, an easily deployable proving service built upon the ZoKrates DSL-toolkit. Initial experiments show that the service can effectively handle workloads larger than 2^{29} in circuit constraints and achieves up to 10x speed-up through parallelization.

II. Analysis

To set the scene, we provide an in-depth analysis of zk-SNARKs applied in VOC. For that, we start with background information on zk-SNARKs, provide an overview of relevant VOC applications, and outline conflicting application workloads and zk-SNARKs limitations. Finally, we give an overview of existing tooling as our related work.

A. Zk-SNARKs in a Nutshell

Zero-knowledge succinct non-interactive argument of knowledge (zk-SNARKs) are a specific type of non-interactive zero-knowledge proof (NIZK) whose short proof sizes and verification times are most suitable to be applied in VOCs. The procedure can be modeled in three operations:

1) The setup ($setup(ecs, srs) \rightarrow (pk, vk)$) where a public asymmetric key pair is generated from the Executable Constraint System (ecs), which encodes the program logic in a provable representation, and a circuit-dependent structured reference string (srs). Proving and verification keys (pk, vk) are both bound to the ecs. The setup builds upon the assumption that the srs is securely disposed to prevent fake proofs.

2) The proving ($P(ecs, x, x', pk) \rightarrow \pi$) is executed in two steps: First, a witness w is generated by executing the ecs on the public inputs x and the private inputs x'. The tuple of (x, x') is commonly referred to as the proof argument. The witness w represents a valid variable assignment of the ecs for the inputs. w together with pk serve as input to the proof generation with the final SNARK π as its output.

3) The verification ($V(\pi, x, vk) \rightarrow \{0,1\}$) is executed on-chain on the proof π and the public inputs x using the verification key vk.

B. VOC Applications

Zk-SNARKs used in VOCs have shown to advance scalability and/or privacy of blockchain-based applications.

1) Oracles provide off-chain data to smart contracts. Since this data is often large, may contain confidential information, or be ill-formatted, a pre-processing of this data on intermediary off-chain nodes is often required, e.g., to bring data into the right format or hide confidential information. zk-SNARKs-based VOCs have been applied to realize this pre-processing in a verifiable manner [4].

2) Rollups enhance blockchain scalability by processing a bundle of transactions off-chain and only sending the computed state to the blockchain. To maintain correctness, on-chain verification ensures the rollup is computed on the latest state with valid transactions. ZkRollups use VOC to make off-chain transaction execution verifiable on-chain. Examples of operational ZkRollup systems are Polygon Hermez [13] and Aztec [14].

3) Relays or bridges enable cross-blockchain interactions. They operate by verifying the consensus rules of the originating blockchain on the receiving ledger, i.e., block headers from the source chain are validated to the target blockchain. In [5], VOCs are applied to validate headers of the source chain off-chain and only verify the zkSNARK-based proof on the target blockchain, thereby, saving transaction costs.

C. Workloads and Limitations

Zk-SNARKs-based proving in VOC applications have the following workload needs in common:

- *Large Workloads*: The more computation is outsourced to the off-chain node, the higher the transaction cost savings on the blockchain. For example, relays and rollups become more economical the more transactions and headers are processed at once.
- *Varying Workloads*: Off-chain provers executing VOC are facing varying workloads. In oracles, for example, data from further sources needs to be pre-processed before being provided to the blockchain. In relays, different numbers of block headers need to be verified depending on the position of the target block.

These application workloads clash with the inherent limitations of zk-SNARKs:

- *Large Overheads*: The succinctness and fast verification of zk-SNARKs is only achieved by offloading most of the computational complexity and memory requirements onto the proof generation process. This results in heavy resource requirements introducing barriers to the adoption of VOC particularly for large workloads.
- *Static Execution Model*: An ecs is a static script of fixed application logic and input parameters. This inflexibility restricts the interoperability with other applications and makes adoptions to varying workloads cost-intensive.

Solving conflicting limitations inherent to zk-SNARKs and application-driven workloads requires suitable tooling.

D. Related Work

Zk-SNARKs rely on advanced mathematics and require a high degree of program-level optimizations to generate proofs efficiently. In recent years, a multitude of projects have been founded to facilitate the development of zk-SNARKs and other NIZK proof variants. Depending on their level of abstraction and performance, these tools fall under three main categories:

1) Libraries: At the lowest abstraction level, libraries like Ark [15], Bellman [16], and Gnark [17] serve as interfaces in popular programming languages, allowing the programming of zero-knowledge primitives for various proving systems [18]. These tools are highly favored due to their customizable nature and allow for the creation of highly optimized circuits for specific proof systems. However, this low level of abstraction requires deep expertise and significant investment in tailor-made solutions with high risks and maintenance costs associated with them, raising the entry barrier and impeding rapid prototyping.

2) DSL-Toolkits: Building on top of libraries, DSL-Toolkits like ZoKrates [3], Circom [8], and Noir [9] provide developers with a set of tools to hide most of the complexity around setups, proving, and verification. At their heart, they consist of a Domain-Specific Language (DSL) as an abstract and high-level interface to define computational problems that can be compiled into ecs, which serves as the foundation for subsequent setup, proving, and verification. While DSL-toolkits facilitate the development of VOC applications with zkSNARKs, they do not provide adequate support for integrating these static and resource-intensive proving scripts into modern application system environments like clouds.

3) Zero-knowledge Virtual Machine (zk-VM)s: Further simplifying the development of ZKPs, zk-VMs like Risc Zero [19], Polygon Miden [20], and eigen-zkVM [21] translate programs written in popular programming languages into VMs with zk-instructions being natively supported in the VM's instruction set [22]. Apart from the outcome of the original program, zk-VMs deliver a ZKP as a proof of correct execution. However, due to the additional number of native instructions they support, current zk-VMs are 10-100 times slower than circuit-based tools [23]. Furthermore, zk-VMs heavily rely on zk-STARKs which have higher verification costs compared to zk-SNARKs.

III. Proving Service

Setting the focus rather on rapid prototyping or circuit optimizations, at their current state, existing tools do not sufficiently support developers to solve the conflict between VOC application workloads and zkSNARKs limitations. Addressing this gap, we introduce a service-oriented approach for VOC that facilitates the use of the cryptographic procedures of zkSNARKs within cloud system architectures. Our system allows for executing ecs as encapsulated application logic in containers that are deployable to different machines realizing

scalability, provide interoperability with other services, and are better manageable, e.g. in that dedicated storage can be assigned. For that, we derive general requirements for proving services, give an overview of the service-oriented system architecture, and describe the internals of the proving service.

A. Requirements

Proving services distinguish from existing tooling by leveraging modern service-oriented system engineering principles. The following should be fulfilled by cloud-native *proving services*:

- *Scalability:* Developers should be able to let the proving service exploit the cloud's horizontal and/or vertical scalability to adapt to varying workloads without compromising performance-related metrics like execution times.
- *Interoperability:* Developers should be able to integrate ecs with other applications and services. The proving service should be able to obtain inputs from different sources and provide proofs to different blockchains.
- *Manageability:* Developers should be technically supported to deploy optimized ecs to cloud execution infrastructures that fit the workloads and requirements at hand. Furthermore, they should be able to observe deployed proving processes and assign necessary resources when required.

B. Service Oriented Verifiable Off-chain Computation

The requirements 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 with the computation's output.

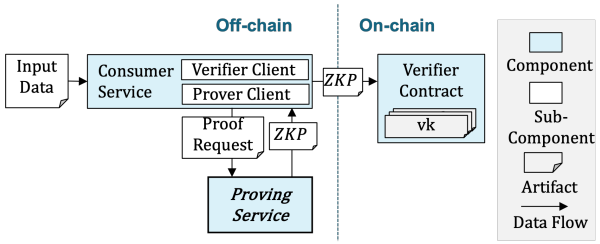

Fig. 1: Proving Service Model

Following the VOC model [3], we distinguish between the blockchain infrastructure hosting the *verifier contract* and an arbitrary 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.

C. Service Architecture

The *Proving Service* depicted in Figure 2 is an application service that runs stand-alone on the prover's off-chain infrastructure. It serves proof requests by exposing the proving-related operations, e.g., i.e. witness computation and proof generation, through an *Application Program Interface* (API) as minimal functionality to the consumer service.

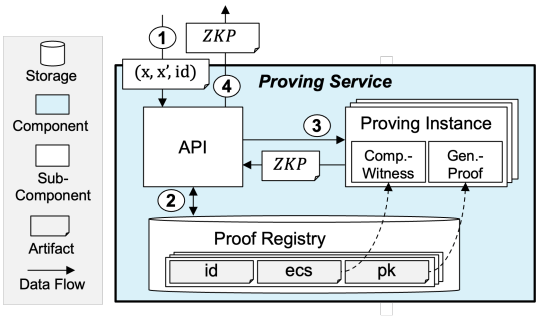

Fig. 2: Proving Service Architecture

The procedure can be summarized in four simple steps: First, the proving service receives the *Proof Request* containing (x, x', id), where x and x' represent the private and public arguments and id represents the identifier of the addressed ecs. Second, using the id the API fetches the corresponding ecs and 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 is computed as an intermediary artifact that represents a valid variable assignment in the ecs for the provided inputs (x, x'), then the witness and the pk serve as input to the ZKP generation. Fourth, the ZKP is returned to the consumer service through the API upon successful execution.

IV. Evaluation

The previous service-oriented architecture serves as a technology-agnostic blueprint for building proving services. For evaluation, we technically instantiate the proving service for ZoKrates [3] and conduct experiments on typical workloads for VOC.

A. ZoKrates-API

To technically realize the proving service, we 'servicify' ZoKrates and present the ZoKrates-API[1] as a ready-to-use open-source software.

We chose ZoKrates as it is one of the most mature projects and allows for plugging different backend libraries including Ark [15] which provides a high-performance and safe-memory library. Furthermore, it supports multiple elliptic curves thereby allowing proof verification on different blockchains, and offers interoperability with other popular DSL-toolboxes like Circom [8].

[1] https://anonymous.4open.science/r/zokrates-api-D176

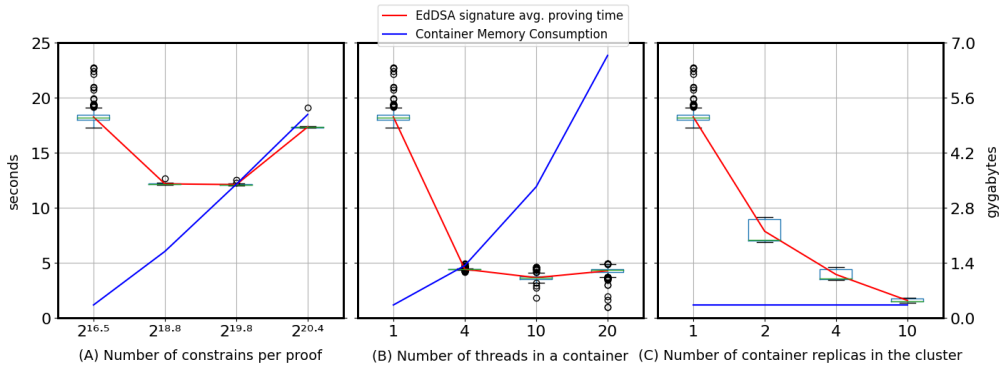

Fig. 3: ZoKrates-API Proving Time and Memory Consumption

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 thereby allowing HTTP requests through the consumer service. For the scope of this paper, the ZoKrates-API only supports the GM17 [24] proving system.

We containerized the ZoKrates-API using Docker [25] making the services easily deployable among a wide range of machines and allowing us to further leverage cloud-native tools like Kubernetes [26] for horizontal scalability, manageability, and observability. Furthermore, the ZoKrates-API supports multi-threading so a single instance can compute multiple proofs in parallel.

B. Experiments

To gain insights into the performance behavior of a cloud-native proving service, we deployed the containerized ZoKrates-API on a Kubernetes cluster running on a machine with 130GB RAM space, a *Xeon Silver 4114* processor with 52000 milicores CPU, and a connection bandwidth of 10Gb/s. As a workload for our experiments, we generated a large number of EdDSA signatures commonly applied in rollups [6] and oracles [4]. The entire dataset mounts for more than 2^{29} of circuit constraints similar to [27].

We conducted three experiments and measured the average proving time per signature in [sec] and the memory consumption in [gb] using different cluster configurations for each experiment. The results for the respective experiments are depicted in Figure 3.

For the first experiment, we split the dataset into signature batches with the following constrain number $\sim 2^{16}$, 2^{18}, 2^{19}, and 2^{20}. The batches series were executed on a container assigned with 8vCPU and 16GB of memory. Figure 3A) shows that a 33% time reduction can be achieved by adjusting the hardware requirements to a given circuit size (or vice versa), whereas the memory consumption increased per batch size.

For the second experiment, we used the same container and kept the smallest batch size but enabled multi-threading allowing for 1, 4, 10, and 20 instances to run in parallel. Figure 3B) shows that significant performance improvements can be gained by allowing parallel computing, however, this approach stagnates quickly.

For the third experiment, we again kept the smallest batch size and exploited horizontal scalability on the container level allowing for 1, 2, 4, and 10 containers running in parallel on the cluster, all single-threaded. Figure 3C) shows that the proving time decreased proportionally as more containers were added whereas the memory consumption per container remained the same.

As these experiments demonstrate, executing proving services in a cloud-native environment using Kubernetes allows for horizontal (more nodes) and vertical (larger nodes) scalability of the proving service and facilitates manageability through the Kubernetes Control Plane, both especially useful for large proving workloads. Furthermore, the containerized ZoKrates-API facilitates interoperability with other services through the standardized interfaces of the service containers.

V. Conclusion

zk-SNARK's proof generation is a costly engineering challenge that greatly hinders the widespread adoption of VOCs. To this end, we propose a context-free service approach, which can be used to design a cloud-native proving service for VOC, as demonstrated in this paper. Our implementation shows that the servicification opens up new strategies to scale up proof generation, while at the same time greatly simplifying the development and deployment of such functionalities.

This paper describes an important first building block for integrating current hot topics in zero-knowledge research with blockchain infrastructure. Recursive proofs, collaborative SNARKs, or computation off-loading all leverage the idea of splitting circuits into smaller pieces and to either distribute or to compute these pieces in parallel. However, all of the previous topics require of a scalable proving service capable to proof arbitrary statements on demand. This paper introduces the proving service concept and a solution for the ZoKrates ecosystem addressing this need.

REFERENCES

[1] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 459–474.

[2] S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-snark parameters in the random beacon model," *Cryptology ePrint Archive*, 2017.

[3] J. Eberhardt and S. Tai, "Zokrates - scalable privacy-preserving off-chain computations," in *IEEE International Conference on Blockchain*. IEEE, 2018.

[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] M. Westerkamp and J. Eberhardt, "zkrelay: Facilitating sidechains using zksnark-based chain-relays," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 378–386.

[6] E. Ben-Sasson. (2020) A cambrian explosion of crypto proofs. [Online]. Available: https://nakamoto.com/cambrian-explosion-of-crypto-proofs/

[7] 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.

[8] 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.

[9] K. Wedderburn. (2023) Noir documentation. [Online]. Available: https://noir-lang.github.io/book/index.html

[10] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orrù, "zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks," Cryptology ePrint Archive, Paper 2023/1503, 2023, https://eprint.iacr.org/2023/1503. [Online]. Available: https://eprint.iacr.org/2023/1503

[11] M. Labs. (2023) The cost of intelligence: Proving machine learning inference with zero-knowledge. [Online]. Available: https://medium.com/@ModulusLabs/chapter-5-the-cost-of-intelligence-da26dbf93307

[12] C. Network. (2023) The pantheon of zero knowledge proof development frameworks (updated!). [Online]. Available: https://blog.celer.network/2023/08/04/the-pantheon-of-zero-knowledge-proof-development-frameworks/

[13] Polygon Hermez Network. (2023) White paper: Scalable payments. decentralised by design, open for everyone. [Online]. Available: https://hermez.io/polygon-hermez-whitepaper.pdf

[14] A. Gabizon, Z. Williamson, and T. Walton-Pocock. (2021) Aztec yellow paper. [Online]. Available: https://aztec.network/

[15] arkworks contributors, "`arkworks` zksnark ecosystem," 2022. [Online]. Available: https://arkworks.rs

[16] bellman contributors, "`bellman` is a crate for building zk-snark circuits," 2023. [Online]. Available: https://github.com/zkcrypto/bellman

[17] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "Consensys/gnark: v0.9.0," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

[18] D. Boneh, S. Goldwasser, D. Song, J. Thaler, and Y. Zhang, "Zero knowledge proofs (mooc, spring 2023):programming zkps," https://zk-learning.org/assets/lecture3-2023.pdf, 2023, accessed 23/10/23.

[19] risc zero contributors, "Risc zero is a zero-knowledge verifiable general computing platform based on zk-starks and the risc-v microarchitecture." 2023. [Online]. Available: https://www.risczero.com/

[20] Polygon, "Miden a stark-based virtual machine." 2023. [Online]. Available: https://polygon.technology/polygon-miden

[21] eigen-zkVM contributors, "eigen-zkvm: a zkvm on layered proof system," 2023. [Online]. Available: https://github.com/0xEigenLabs/eigen-zkvm/

[22] T. Dokchitser and A. Bulkin, "Zero knowledge virtual machine step by step," Cryptology ePrint Archive, Paper 2023/1032, 2023, https://eprint.iacr.org/2023/1032. [Online]. Available: https://eprint.iacr.org/2023/1032

[23] W. D. . T. Chung. Zk10: Analysis of zkvm designs. Youtube. [Online]. Available: https://www.youtube.com/watch?v=tWJZX-WmbeY

[24] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Annual International Cryptology Conference*. Springer, 2017, pp. 581–612.

[25] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[26] "Kubernetes Manual," https://kubernetes.io/docs/reference/, 2023, [Online; accessed 11-NOV-2023].

[27] A. Chiesa, R. Lehmkuhl, P. Mishra, and Y. Zhang, "Eos: Efficient private delegation of zksnark provers," in *USENIX Security Symposium. USENIX Association*, 2023.