zkSNARKs Libraries for Blockchains: a Comparative Study

Domenico Tortola[1], Andrea Pelosi[1,2], Giuseppe Gabriele Russo[1], Paolo Mori[3,*] and Laura Ricci[1]

[1]*University of Pisa, Pisa, Italy*

[2]*University of Camerino, Camerino, Italy*

[3]*Consiglio Nazionale delle Ricerche - Istituto di Informatica e Telematica, Pisa, Italy*

Abstract

Blockchain technology is currently being used in a large number of application scenarios besides the cryptocurrency exchange one, mainly thanks to the introduction of smart contracts, which allow to implement applications that are executed on the blockchain (Decentralised applications). Smart contracts' code and data are visible by all the participants to the blockchain, thus preventing the adoption of blockchain technology in those application scenarios where data privacy is required. To address this problem, Zero Knowledge Succint Non-interactive Argument of Knowledge (zkSNARK) proofs have been proposed, which allow smart contracts to verify a known condition on secret data without revealing it. To integrate the zkSNARK technology in their smart contracts, developers can take advantage of two popular libraries: Circom and Zokrates. However, when choosing which of the two to adopt, developers should take into account the cost in terms of gas and storage space of the resulting code. To this aim, this paper contributes by performing an experimental comparison of the two libraries. In particular, three well know problems requiring data privacy have been selected, the smart contract implementing the corresponding privacy preserving verification of a known condition on secret data have been produced exploiting the two libraries, and the related performance in terms of smart contracts deployment and execution costs and storage space required for the zkSNARK data (circuits, proofs and keys) have been measured, compared, and discussed.

Keywords

Blockchain, Privacy, Zero Knowledge, zkSNARK

1. Introduction

In the last years, blockchain technology is having an ever increasing spread both in research organizations and business companies. As a matter of fact, the introduction of smart contracts allowing to create Decentralised Applications (dApps) brought to the development of blockchain based solutions in many distinct application scenarios besides the cryptocurrency exchange one, such as: electronic voting, decentralized notary, supply chains management, identity management, access control, healthcare records management, and many others.

One of the main features that contributes to the success of the blockchain technology is transparency, i.e., the information (code and data) stored on a blockchain can be seen by all the participants. If, on the one hand, this feature allows all participants to potentially access and verify all the transactions on the blockchain, on the other hand it does not allow to preserve the privacy of the data stored on the blockchain and used for smart contracts execution, thus preventing the adoption of blockchain technology in many applications.

For instance, let us suppose that the University of Pisa wants to grant the right of executing a given smart contract S for conducting the experiments for the Distribute Ledger Technology course to the students who have paid a predefined fee or have a low income. To allow S to decide whether to grant the access to a student invoking it, the University of Pisa (or another trusted actor) could register the student's income or the fee payment on the blockchain, but this solution does not preserve student's

*Corresponding author

domenico.tortola@phd.unipi.it (D. Tortola); andrea.pelosi@phd.unipi.it (A. Pelosi); g.russo55@studenti.unipi.it (G. G. Russo); paolo.mori@iit.cnr.it (P. Mori); laura.ricci@unipi.it (L. Ricci)

0009-0003-4295-0960 (D. Tortola); 0000-0002-0877-7063 (A. Pelosi); 0000-0002-6618-0388 (P. Mori); 0000-0002-0877-7063 (L. Ricci)

privacy because it would disclose the student's income. Hence, a solution allowing S to evaluate conditions on secret values without disclosing them is required.

This is where Zero Knowledge Succint Non-interactive Argument of Knowledge (zkSNARK) proofs [1, 2] come into play. As a matter of fact, the zkSNARK technology allows to write smart contracts able to verify a known condition on secret data without revealing the latter. This opens the way to a large number of blockchain based applications where data privacy is a critical requirement. For instance, the authors of [3] exploit the zkSNARK technology to implement smart contracts evaluating Attribute Based Access Control (ABAC) policies [4] where the outcome of the policy evaluation is published on the blockchain (i.e., access granted/denied) without actually disclosing on the blockchain the values of the attributes used for the policy evaluation.

The easiest way for developing a smart contract based application exploiting the zkSNARK technology is exploiting an existing library, being Circom and Zokrates among the two most popular ones. This would relieve the developers from learning the mathematical foundations the zkSNARK technology is based on, since they should simply need to learn the specific languages used by the two libraries to express the conditions that will be evaluated on secret data. However, when choosing which of the two libraries to adopt for their dApps, developers should take into account the cost in terms of gas and storage space of the resulting code.

In the light of the previous considerations, the contribution of this paper is an evaluation and comparison between the two popular libraries implementing the zkSNARK protocol, Circom and Zokrates. In particular, such evaluation has been executed by implementing three well known problems where a privacy preserving condition involving secret inputs must be verified on the blockchain, namely the age verification, the Sudoku, and the Hamiltonian cycle problems, and by *i)* computing the gas cost of deploying and executing the corresponding smart contract, and *ii)* evaluating the size of the proofs, produced keys, and circuits varying the dimensions of the problems. These experimental results are then discussed to provide a kind of guideline for developers helping them to understand which library is more convenient to be used in their specific application scenarios.

The paper is structured as follows: Section 2 recalls the relevant background on blockchain and zero knowledge proofs, Section 3 briefly surveys relevant research in the area of zkSNARKs for blockchain, Section 4 introduces the Circom and ZoKrates libraries in more details. We perform an experimental comparison between the two libraries in Section 5 and discuss our findings in Section 6. We sum up the main insights and conclude our paper in Section 7.

2. Background

2.1. Blockchain technologies

A blockchain is a data structure made by data blocks, usually storing data in form of transactions, connected each other by hash pointers. This transaction ledger is shared among a peer-to-peer network, with every node holding a copy of the entire data structure. Users in the network execute transactions, which are grouped in a block and appended to the ledger after the block is validated. The block validation is performed through a consensus algorithm, with Proof-of-Work (PoW) [5] and Proof-of-Stake (PoS) [6] as the most used. After the validation, every node updates their copy of the ledger appending the new block.

Blockchains were popularized by Bitcoin [7], which built a decentralized network dedicated to digital currency trading and secured by cryptography, while newer projects such as Ethereum [8] focused on blockchain based application execution introducing smart contracts, blocks of code executed automatically executed on the blockchain as a transaction execution.

2.2. Zero Knowledge proofs

Zero Knowledge proofs (ZKPs) are a broad class of cryptographical constructs first introduced by Goldwasser et al. in [9]. A Zero Knowledge Proof involves two actors: a prover and a verifier. The former aims to prove the validity of a statement to the latter, without leaking additional information besides the validity of the statement being proved. Hence, ZKPs on the one hand allow to keep some data secret, while, on the other hand, allow to demonstrate that a statement based on such data is verified. For ZKP systems, three properties hold:

- *Completeness:* if the statement is true, a honest prover will always be able to convince the verifier except with negligible probability;
- *Soundness:* if the statement is false, a malicious prover will be able to convince a verifier at most with negligible probability;
- *Zero Knowledge:* if the statement is true, the verifier (even if malicious) will not learn anything about the statement from the proof besides the fact that the statement is true.

One common example to understand ZKPs is the well known Alibaba cave [10]: in this scenario, there is a ring shaped cave with only one entrance and a magic door at the other side of the ring, which can only be opened using a magic word. A prover, named Peggy, knows the secret magic word and a verifier, named Victor, challenges Peggy to prove the knowledge of the magic word. Therefore, Peggy has to provide to Victor a proof about the knowledge of such word without revealing it. To provide such proof of knowledge, the two play a game: Peggy will enter the cave and picks a side to proceed, that could be left or right, say left. Victor, who do not know which side Peggy picked, will ask to Peggy to come out from the cave from a specific side. If Victor picks the left side as exit, Peggy will simply walks back. If Victor picks the right side instead, Peggy has to use the magic word to open the door and walk over the entire ring to come out from the right side. In both cases, Peggy will be able to pick the correct exit side. This single game could not be sufficient to convince Victor about the knowledge of the word: there is a 50% chance that Peggy simply entered the correct side from the start, not using the magic word at all. The game can be repeated multiple times, reducing game by game the possibility that Peggy is simply lucky, until Victor is convinced.

2.2.1. zkSNARKs

zkSNARKs (*Zero Knowledge Succint Non-interactive Argument of Knowledge*) [2] are a very popular class of ZKP protocols. In a zkSNARK setting, an efficient (i.e. polynomially bounded) prover wants to prove the knowledge of a *witness* for a statement to a verifier in a complete, knowledge-sound and zero-knowledge way. Knowledge soundness is a stronger notion of soundness which states that a (malicious) prover that does not know a witness for a given statement can not convince a verifier about the validity of the statement. To model this, a knowledge extractor is employed, i.e. an algorithm that can compute a witness whenever a (malicious) prover provides a valid argument. The notion of “Argument" refers to the fact that the knowledge-soundness property should be satisfied only for a poly-bounded prover, in contrast with the stricter notion of “Proof", in which the prover has unbounded computational resources, see [11, 12] for details.

Besides the properties previously mentioned, a zkSNARK system guarantees two additional properties: *succintness*, meaning that the proof has a small size and can be verified efficiently, and *non-interactivity*, meaning that constructing and verifying a proof requires little interaction between the prover and the verifier or no interaction at all.

zkSNARKs life cycle is made by two main phases: *arithmetization* and *commitment*. In the arithmetization phase, the statement to prove is modeled as an arithmetic circuit, that mathematically maps how input values produces output values. One or more constraints can be defined on the model, to guarantee the correctness of the proof. The final output of the arithmetization is a polynomial representation which accounts both the model and the constraints. On the other hand, in the commitment phase the prover uses cryptographical schemes to generate values (the *commitments*) that will be used in the verification process to guarantee the zero knowledge property. Usually, commitments are generated starting from the coefficients of the polynomial produced in the arithmetization phase applying hash functions to hide the real values.

zkSNARKs usually require a preliminary trusted setup phase between the prover and the verifier, in which they generates two keys: a proving key, which is known only to the prover and will be used to generate the proof, and a public verification key which can be used by the verifier to verify the proof. The setup phase produces some data, often refereed as "toxic waste", which has to be destroyed right after the key generation to maintain the security of the protocol. If the trusted setup phase is executed in a decentralized way, it is sufficient that one of the participants to the setup phase is honest and deletes their part of the toxic waste.

3. Related works

The integration of zkSNARKs in blockchain based applications is a very promising and active research line, both in literature and in enterprise applications. zkSNARKs are applied to blockchain to both Layer 1 blockchains and Layer 2 applications. As Layer 1 application, we mention ZCash [13, 14], a blockchain platform that implements zkSNARKs to allow users to execute private transactions. In particular, a zkSNARK is generated to prove the validity of a private transaction to the network users not directly involved in it, in order to make it possible to validate the transaction without reveling any other information except of the transaction validity. On the other hand, there are several applications of zkSNARKs in the context of blockchain Layer 2 applications, aimed to provide scalability and/or privacy when required.

Simunic et al. in [15] survey the methods to verify computations, focusing on blockchain applications. The authors highlight how ZKPs in general can be a powerful tool to implement verifiable computing for Layer 2 applications, ensuring at the same time a high level of privacy.

Partala et al. [16] describe the applications of general non-interactive ZKPs to blockchain, with particular attention to how to obtain confidential transactions and privacy-preserving smart contracts through ZKPs. The authors describe numerous protocols and techniques for implementing ZKPs and present several blockchain based applications exploiting them.

Kosba et al. in [17] propose a framework to build privacy-preserving smart contracts, in which users inputs and application logic are executed off-chain and a zkSNARK, related to the off-chain result, is verified by the on-chain smart contract.

In terms of blockchain applications, zkSNARKs are used with success in ZK rollups [18], a class of Layer 2 applications in which zkSNARKs are used as validity proofs paired with an off-chain computation, to assess the validity of the result of the computation, committed on the Layer 1 blockchain.

Furthermore, there is also a growing interest in benchmarking of zkSNARKs libraries, similarly to what we will discuss in this paper. Ernstberg et al. in [19] present a framework to evaluate, in terms of memory consumption and execution time, several zkSNARK protocols. The benchmark of zkSNARK libraries was also studied by Baghery et al. [20], which benchmark the setup phase in a subset of updatable ZK protocols, and by Steidtmann et al. [21] which focus on benchmarking several Circom circuits, mostly involving hash functions. A similar work, not restricted to Circom circuits, can be found in [22]. However, our work proposes a different point of view on the challenge of benchmark zkSNARK libraries and protocols with respect to the mentioned related works. More specifically, our focus is related to the blockchain applications of such libraries, being then less general-purpose with respect of the cited works. Moreover, we focused on a wider range of evaluated scenarios, not limited to cryptographical operations. Finally, the metrics we evaluated in our experiments are not limited to the memory evaluations or the time consumption, but we provided the evaluation of several other metrics about the smart contract related to an on-chain proof verifier.

4. Implementing zkSNARK protocols on Ethereum: Circom and ZoKrates

There are are several ways to implement zkSNARK protocols that produce proofs verifiable on Ethereum. Among them, in this paper we focus on two of the most popular zkSNARK libraries: Circom and

Zokrates. The motivations behind this choice are multiple: first, the two libraries are widely used to generate zkSNARKs in a various range of different applications, including blockchain related ones. The integration with blockchains is the main motivation driving our selection, since both libraries are capable to generate a verifier smart contract alongside each proof. Finally, the fact that the two libraries have several common traits but at the same time crucial differences, as explained in the rest of this section, make the comparison even more significant. The rest of this section describes the two libraries.

4.1. Circom

Circom [23] is a library composed by a domain-specific language, which can be used to code the statement to prove, and a compiler which transforms the code into an arithmetic representation known as *Rank 1 Constraint System* (R1CS). To execute the trusted setup, Circom integrates SNARKJS as an external library. Also, the witness has to be computed externally, using the R1CS (plus some extra files produced compiling the circuit, according to the chosen compilation options) as input. This operation can be executed using NODEJS or C++ wrappers. Circom language allows developers to define the problem as a sequence of inputs (*signals*) and operations between them (*wires*), such that the transformation into an arithmetic circuit is optimized since the development stage. Furthermore, the language allows to distinguish between public and private inputs. Finally, the library supports the proof verification executed by a dedicated Solidity smart contracts.

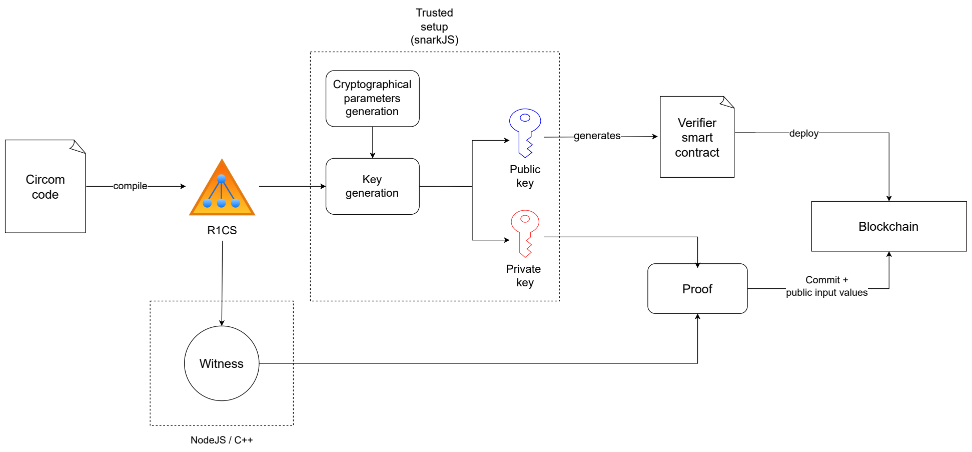

Figure 1: Circom workflow overview

Figure 1 provides an overview about how Circom works. First, the Circom code is compiled into the corresponding R1CS representation. An R1CS is a system of polynomials with degree 1, which combines the arithmetic operations composing the circuit that are needed to prove the initial statement and the constraints defined on the circuit. In Circom, the trusted setup is delegated to snarkJS, which executes two phases: in the first phase, independent by the circuit, the CRS is generated executing the Power of Tau [24] ceremony. In the second phase, the CRS and the R1CS are used to generate the key pair needed to generate the proof (private key) and to verify it (public key). The R1CS is also used to generate the witness: this operation is executed by an external component as well, in particular using a NodeJS or a C++ dedicated module. The witness and the private key are used to generate the proof, while the public key is used as input to generate the Solidity smart contract. The smart contract can be deployed on any EVM based blockchain, since it is written in Solidity, to enable the on-chain verification process.

4.2. ZoKrates

ZoKrates [25] is a zkSNARK library that provides a domain specific language to write programs defining the statement to prove and the necessary inputs, a compiler to compile the mentioned programs into arithmetic circuits, and a procedure to generate proofs and to verify them. The programming language is an high-level language, making easier for developers to define the statement logic. Also, the language allows the distinction of inputs into public and private. The compiler compiles ZoKrates programs into an arithmetic representation called *ZoKrates Intermediate Representation* (ZIR).

A trusted setup is required to generate the private and public keys which are used, respectively, to generate and verify the proofs. Finally, ZoKrates is able to generate tailor-made Solidity smart contracts to verify proofs on EVM based blockchains, such as Ethereum.

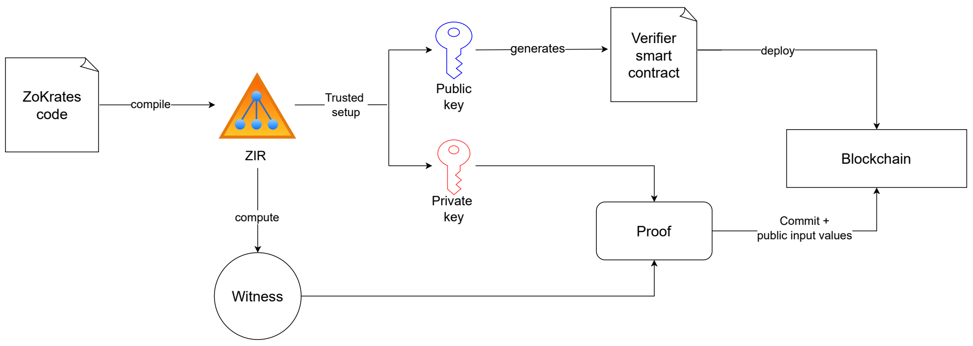

Figure 2: ZoKrates workflow overview

Figure 2 illustrates the ZoKrates workflow. The ZoKrates code, written using the dedicated language, is compiled into a ZIR by the compiler. The ZIR is an extended and more abstracted version of the R1CS representation, which optimizes both the program execution and the proof generation. The ZIR contains constraints (conditions to generate the arithmetic representation) and directives (used by the interpreter). ZoKrates supports a locally executed trusted setup, which takes in input the ZIR file and outputs a long proving key and a short verification key. The ZIR is also used to define the witness, which includes all the data needed to generate the proof, both public and private. This witness, in combination with the private proving key, is used to generate the proof. On the other hand, the verification key is used to generate a Solidity smart contract, which is capable of verifying the proof generated by the corresponding program. Such smart contract can be deployed on any EVM based blockchain to enable on-chain proof verification.

5. Libraries comparison

Exploiting the Circom or the Zokrates library is an easy way for dApps developers for integrating the zkSNARK technology in their dApps to protect data privacy, because they simply have to use the specific constructs provided by the languages of the two libraries to express the conditions that needs to be evaluated on secret data. However, when choosing the library that best fits their dApps, developers must consider the cost in terms of gas and storage space of the resulting code.

For this reason, in this section, we consider how ZoKrates and Circom compare with each other in three different scenarios. We assume that a Layer-2 dApp executes off-chain computation and, after that, publishes on the blockchain a zkSNARK proof of the result so that a dedicated on-chain smart contract can verify it. For a comprehensive comparison, we implement every scenario with both libraries, evaluating both the prover and the verifier side.

Our goal is to provide both the research and the development communities with guidelines that could help in choosing the most suitable zkSNARK library when building privacy preserving dApps. We aim to show how the two libraries behave in different contexts highlightning their strenghts and weaknesses so that developers can make an informed choice, according to their needs. For each scenario we consider, we chose to evaluate the storage consumption by the proving setup (arithmetic circuit file size, proving and verification key size, proof size) and the verifier smart contract deploy and execution costs.

5.1. Scenarios and experimental results

To conduct a thorough comparison, we evaluated the two libraries against a set of problems with increasing complexity with the goal of exploring the features offered by Circom and ZoKrates. We defined three simple scenarios such that we can exploit several features of the programming language provided by the two libraries and how the usage of the main syntactical contructs impact on metrics like the circuit complexity, keys and proof size and smart contract costs, in order to provide the most possible detailed insights about how the libraries handle problems of increasing complexity. Each scenario that we considered in the analysis explores the usage of chosen language features, as Table 1 summarizes. This way, we are able to isolate the impact that the language features have over the metrics we are considering.

Scenario	Context	Evaluated feature
Age evaluation	Proving to be older than a certain threshold	Numeric values and boolean conditions evaluation
Sudoku solution	Proving the knowledge of a solution for a Sudoku grid	Conditional and iterative constructs evaluation
Knowledge of an Hamiltonian cycle for a graph	Proving the knowledge of a valid Hamiltonian cycle for a graph	A more complex test, putting the pieces together

Table 1
Overview of the studied scenarios

For each scenario, we developed the actual proof of knowledge and the corresponding smart contract that acts as the verifier both using Circom and ZoKrates. We used Groth16 [26, 27], a proving scheme based on elliptic curve pairings, for both Circom and ZoKrates. The code of experiments, including both the circuit coding for the two libraries and smart contract codes, is publicly available on GitHub[1]. The experiments were performed on a commodity laptop with Intel Core i5 dual-core 1,6 GHz CPU and 8 GB RAM 2133 MHz LPDDR3. In the rest of this section, detailed descriptions and experimental evaluations for each of these scenarios will be presented and discussed.

5.1.1. Age evaluation

In real world scenarios, the access to services is regulated by policies, mostly defined by laws and regulations. Such policies usually define the requirements that users need to satisfy to access the services, and they can be based on any parameter. For instance, the purchase of a wide range of services is limited by age: the access to sport bets, the possibility to vote, and many others. Therefore, users have to prove to be older than a certain age threshold, which is set according to the required service or the law, to access the service or to purchase the product.

In our tests, a prover wants to prove to be older than a certain threshold, set up by the verifier, without disclosing their real age. Therefore, the age of the prover is considered as the witness, while the age threshold represents the public input.

This first experiment, which is indeed very simple, aims to exploit the basic components of the programming languages provided by the two libraries, i.e., numeric variables and boolean conditions evaluation.

Table 2 reports the size of the files produced by the proof generation process. As expected, since we are dealing with a simple problem, the file sizes are overall very small, with Circom files being even smaller than the ZoKrates ones. No file exceeds the size of few KB, with the circuit file size registering

[1]https://github.com/Giusgarus/zkSNArK-Protocols-Implementation [Online, accessed on 23 April 2024]

	Circuit file size (KB)	Proof size (KB)	Proving key size (KB)	Verification key size (KB)
Circom	4	0,802	8	4
ZoKrates	139	0,849	11	4

Table 2
Storage occupation of age evaluation proof and verification components (in KB)

the biggest difference between the two libraries, with 4 KB for the Circom circuit and 139 KB for the ZoKrates circuit. The other files, instead, have a similar size for the two libraries.

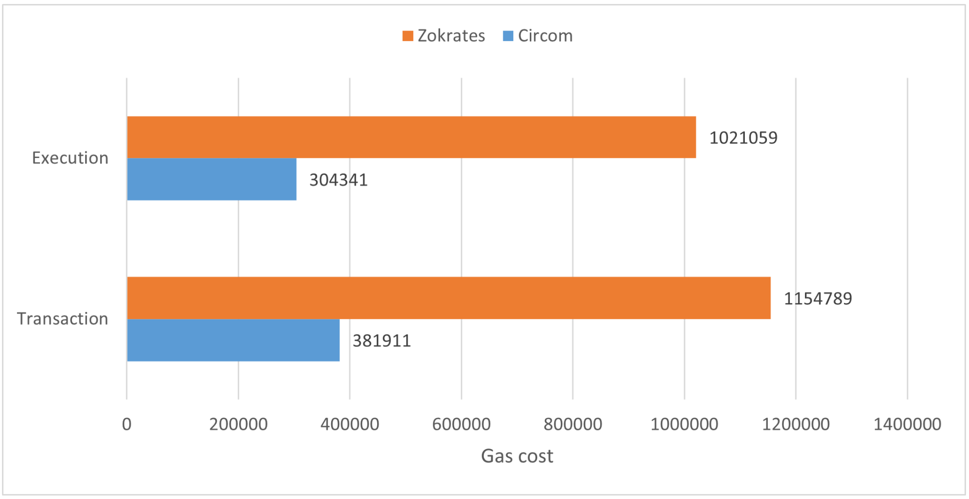

Figure 3: Smart contract costs (deploy and execution) for the age verifier smart contract

Figure 3 illustrates the costs incurred to deploy and invoke the smart contracts generated by Circom and ZoKrates to verify the corresponding proofs. As the graphic highlights, the Circom smart contract appears to be considerably cheaper than the one produced by ZoKrates.

5.1.2. Sudoku solution

Sudoku [28] is a popular puzzle game structured as a square $n \times n$ grid, with $n = 9$ in the most common configuration of the game. The objective of the puzzle is to fill the grid using numbers from 1 to n, such that every number is present only once in every every row, column and $\sqrt{n} \times \sqrt{n}$ square. The starting grid is filled only partially.

In our tests, the prover wants to generate a proof assessing the knowledge of a valid solution for a certain Sudoku grid, maintaining such solution hidden to the verifier. Hence, the witness is the Sudoku solution, while the public input is the starting grid.

The goal of this experiment was to evaluate how the libraries implement the most used syntactic constructs common to programming languages, i.e., conditional blocks ($if/else$) and cycles ($for/while$).

For our experiments, we evaluated 5 different instances of the Sudoku grid, progressively increasing the grid size. More specifically, we considered $n = 4, 9, 16, 25, 36$.

Table 3 summarizes our results about the size of the files produces by the proof generation process for the different Sudoku grids by both libraries. As shown by the table, the circuit files sizes and the proving keys sizes are the metrics where Circom and Zokrates differ the most. For instance, in case of a grid of 36×36, Circom's Circuit file (R1CS file) has a size of 17,3 MB, while the ZoKrates file has size 1,59 GB. The difference is also very pronounced on the proving key size, where the Circom proving key is 57 MB while the corresponding ZoKrates proving key occupies 2,64 GB.

Sudoku grid size		Circuit file size (MB)	Proof size (MB)	Proving key size (MB)	Verification key size (MB)
4×4	Circom	0,006	0,004	0,733	0,008
	ZoKrates	3,6	0,004	7,4	0,004
9×9	Circom	1,1	0,004	4,3	0,02
	ZoKrates	23,3	0,008	17,6	0,016
16×16	Circom	3,4	0,004	11	0,053
	ZoKrates	109,6	0,02	183	0,045
25×25	Circom	8,3	0,004	27	0,119
	ZoKrates	453	0,047	732,6	0,104
36×36	Circom	17,3	0,004	57	0,242
	ZoKrates	1.590	0,098	2.640	0,217

Table 3
Storage occupation of Sudoku proof and verification components (in MB)

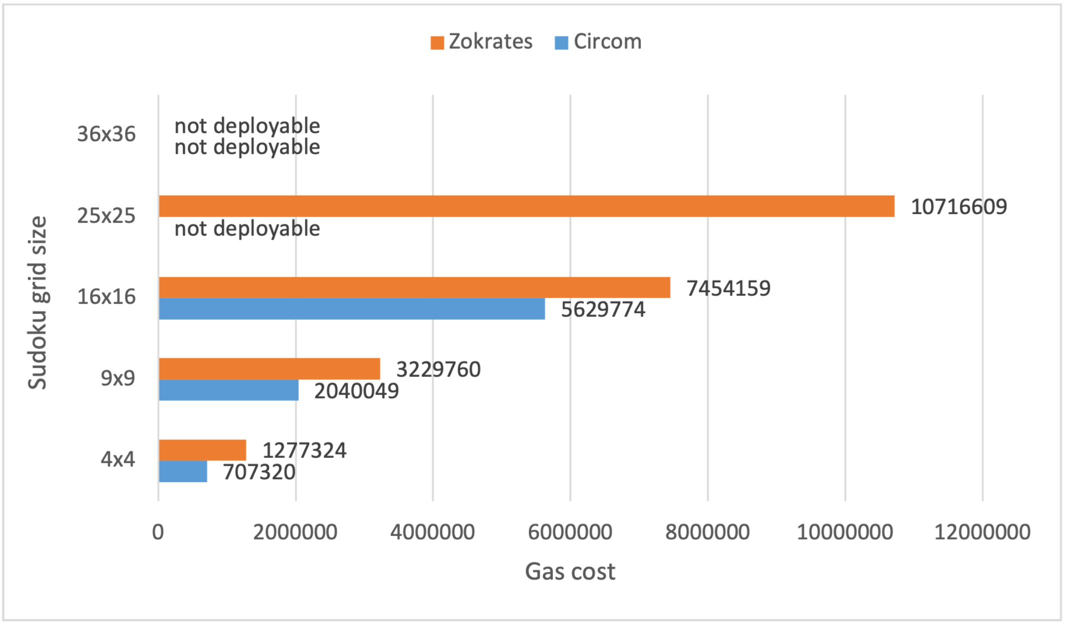

Figure 4: Deploy costs of the Sudoku solver verifier smart contract

Figure 4 shows the cost to deploy the verifier smart contracts generated by Circom and ZoKrates. The first thing to notice is that for big grids the libraries have issues in generating a smart contract to verify the proof of knowledge. This is because Ethereum imposes 24.576 bytes as a maximum size for a smart contract file; a limit that is exceeded by the Circom smart contracts for the 25×25 and 36×36 grids and by ZoKrates contracts for the 36×36 grid, resulting in an exception during the smart contract generation thus making not possible to deploy the corresponding smart contract. For instance, the 36×36 Sudoku configuration requires two 36×36 grids (one for the starting grid and one as the solution) in input, namely 2.592 integers. When the smart contracts were deployable for both Circom and Zokrates, the deployment cost of the ones generated by Circom is always considerably cheaper.

The same trend is confirmed on the execution costs, i.e., the costs for executing the smart contract to verify the proof on the blockchain, as shown in Figure 5. Hence, the execution of the Circom-produced code results cheaper than the execution of the Zokrates one.

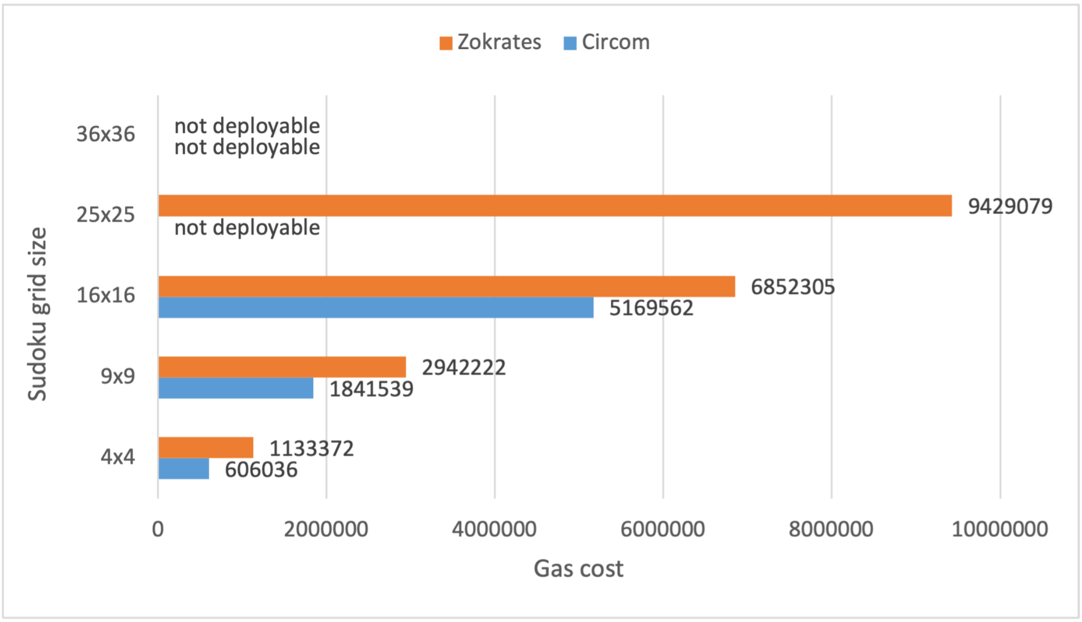

Figure 5: Execution costs of the Sudoku solver verifier smart contract

5.1.3. Hamiltonian cycle

Given a(n undirected) graph, an Hamiltonian cycle [29] is a closed path in the graph that visit every node only once. Hamiltonian cycles are applied in several disciplines, such as computer graphics, genomic mapping, electronic circuit design and more others.

In our tests, a prover generates a proof to convince a verifier that they know a valid Hamiltonian cycle (which will be the witness) for a given graph (which is instead the public input). We defined this final scenario to test the two libraries in a more realistic setting, where both the primitives constructions of the language and the calls to external libraries are executed in the same program, making it possible to evaluate the two libraries in their entirety.

The graphs for this experiment were built in such a way that, for any two nodes i and j, the edge (i, j) exists with probability $p = 0, 5$ (then, the produced graphs were adapted accordingly, so that an Hamiltonian cycle always exists). We considered graphs consisting of 5, 8, 10, 15 and 50 nodes. For conducting our experiments, we passed as inputs the adjacency matrices of the graphs (for an n-node graph, its adjacency matrix has $n \times n$ entries) and the (solution) hamiltonian cycles as arrays of n elements.

Table 4 illustrates the storage occupation resulting from the implementation of the proving workflow for each of the mentioned scenarios. This results follows the trend observed by in the previous experiments, with Circom implementation producing smaller circuit files and proving keys for every scenario instance. More specifically, the size of the Circom circuits increases with the size of the graph, and for 50 nodes, the Circom circuit file size is 401 KB, while for ZoKrates the file is about 100 MB. Regarding proving key, its size increases with the dimension of the graph as well, and we have that, for a graph of 50 nodes, the Circom implementation proving key size is 3,2 MB, against the 106 MB of the ZoKrates one. Also Proofs and Verification Keys sizes increase with the graph dimension, and the former are lower in the Circom implementation, while the latter are slightly lower in the Zokrates implementation. However, the differences in size for Proofs and Verification Keys are considerably less significant than the ones for Circuit Files and Proving Keys.

Figure 6 and Figure 7 both highlight a cost behaviour similar to what happens in the Sudoku scenario: for Circom, smart contract deployment and execution costs are smaller than Zokrates' ones. Also, there is a deployment failure for the 50-node graph verifier contract: it would require to process a total of 2.550 inputs (2.500 for the 50×50 adjacency matrix and 50 for the solution), and the contract size

Graph nodes		Circuit file size (MB)	Proof size (MB)	Proving key size (MB)	Verification key size (MB)
5	Circom	0,033	0,004	0,131	0,008
	ZoKrates	1,2	0,004	2,4	0,008
8	Circom	0,053	0,004	0,262	0,016
	ZoKrates	2,4	0,008	4,6	0,012
10	Circom	0,066	0,004	0,262	0,025
	ZoKrates	3,2	0,008	5,9	0,018
15	Circom	0,139	0,004	0,467	0,045
	ZoKrates	6,3	0,02	11,6	0,041
50	Circom	0,401	0,004	3,2	0,463
	ZoKrates	100,7	0,188	160,6	0,411

Table 4
Storage occupation of Hamiltonian cycle knowledge proof and verification components (in MB)

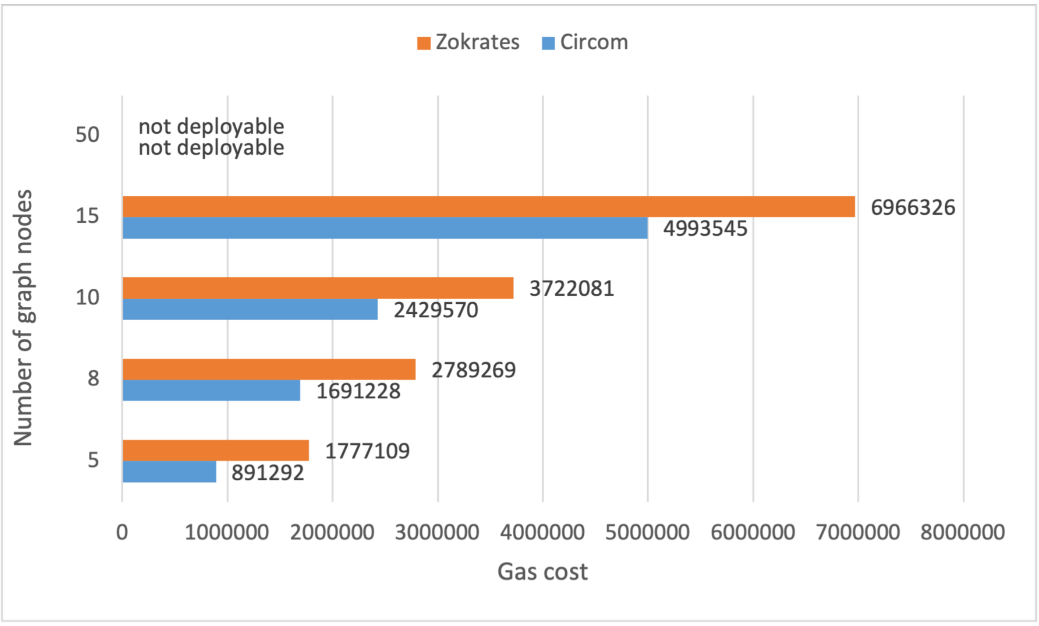

Figure 6: Deploy costs for the Hamiltonian cycle calculation smart contract verifier

would exceed the maximum size supoported for a single smart contract (like in the 36×36 Sudoku scenario).

6. Discussion

In this section, we discuss the insights provided by the experimental results presented in the previous section. From a quantitative point of view, the results show that the dimension of the circuit file and the proving key size grow with the size of the input instances considered. This growth is steady for both Circom and ZoKrates in the scenarios we considered; even though for small inputs Circom and ZoKrates achieve comparable performances (in terms of circuit file size and proving key size), for bigger inputs Circom outperforms ZoKrates since it manages to keep the files size lower. For instance, in our experiments the size of the Circuit files produced by Circom is in the order of tens of MB, while for

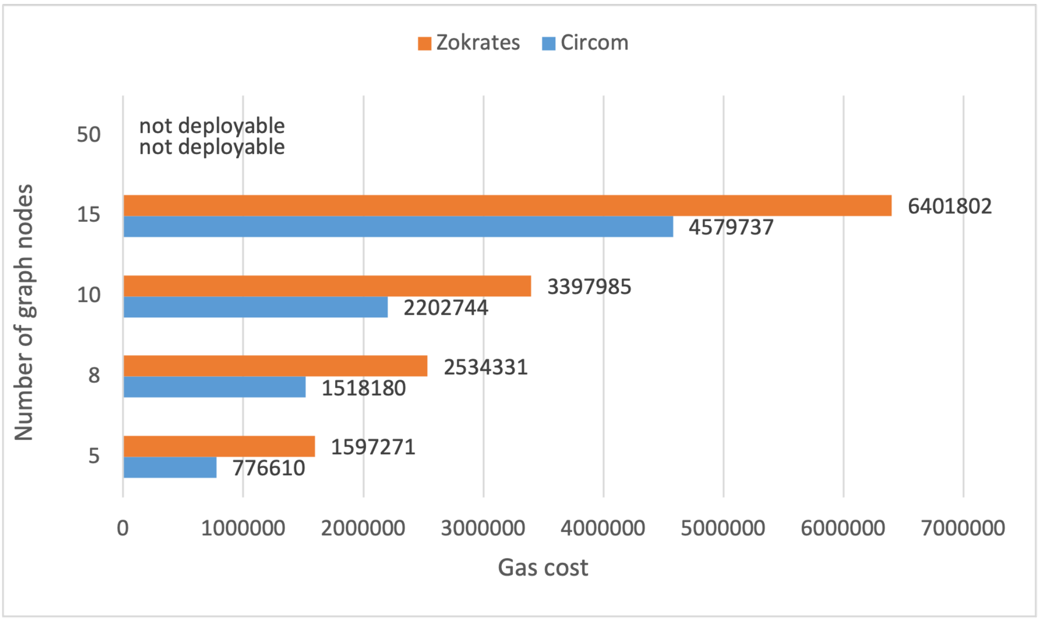

Figure 7: Execution costs for the Hamiltonian cycle calculation smart contract verifier

ZoKrates the size of the Circuit files sometimes reach the order of GB. This suggests that the Circom library performs better when the input size grows significantly, making it more suitable for managing situations where off-chain storage occupation matters, for instance when IoT devices, that usually have limited storage capabilities, are used.

Regarding the actual Proof size and the Verification Key size, in our experiments both Circom and ZoKrates manage to keep the storage occupation low (under 1 MB) without a library that clearly outperforms the other: we argue that this would not have been relevant even if this was the case since the storage occupation of the proof and the verification key files is negligible compared with the storage occupation of the Circuit and the Proving Key files.

For what concerns the on-chain costs, we observe a similar behavior, with Circom smart contracts generally cheaper than the ZoKrates ones. However, as highlighted by the Sudoku solution scenario, ZoKrates optimizes better the smart contract generation, since it managed to compile the contract to handle 25×25 Sudoku grids while Circom did not.

Although the quantitative experimental results show that Circom performs better than ZoKrates, the latter turns out to be more developer friendly, having a higher-level programming language if we compare it with the Circom one. We argue that a higher-level (and thus, easier to use) programming language not only provides a (non-negligible) advantage in terms of development experience, but helps to prevent bugs and inefficiencies that could possibly come up when implementing dApps that perform non-trivial tasks.

Figure 8 gives a view about the coding experience using the Circom and ZoKrates languages, considering only the lines of code needed to define the circuits. We did not take into account the smart contracts code because it is generated automatically, and then not coded by the developer. Considering the implementation of the three analyzed scenarios, the figures shows that coding in ZoKrates requires a number of effective lines of code sensibly lower than coding in Circom. This is particularly appreciable on the Sudoku solution case, where ZoKrates requires less than half lines of code with respect of the counterpart Circom program to achieve the same result (80 for the ZoKrates program, 193 for the Circom one). Furthermore, we noticed that Circom codes need of import several external modules to perform operations, such as comparisons between values. On the other hand, ZoKrates appears to be more independent in this sense, not requiring any import in none of the studied scenarios.

Additionally, ZoKrates manages the entire proof generation workflow without any crucial external dependency (such as snarkJS, that is necessary for Circom's workflow instead), and that could lead to

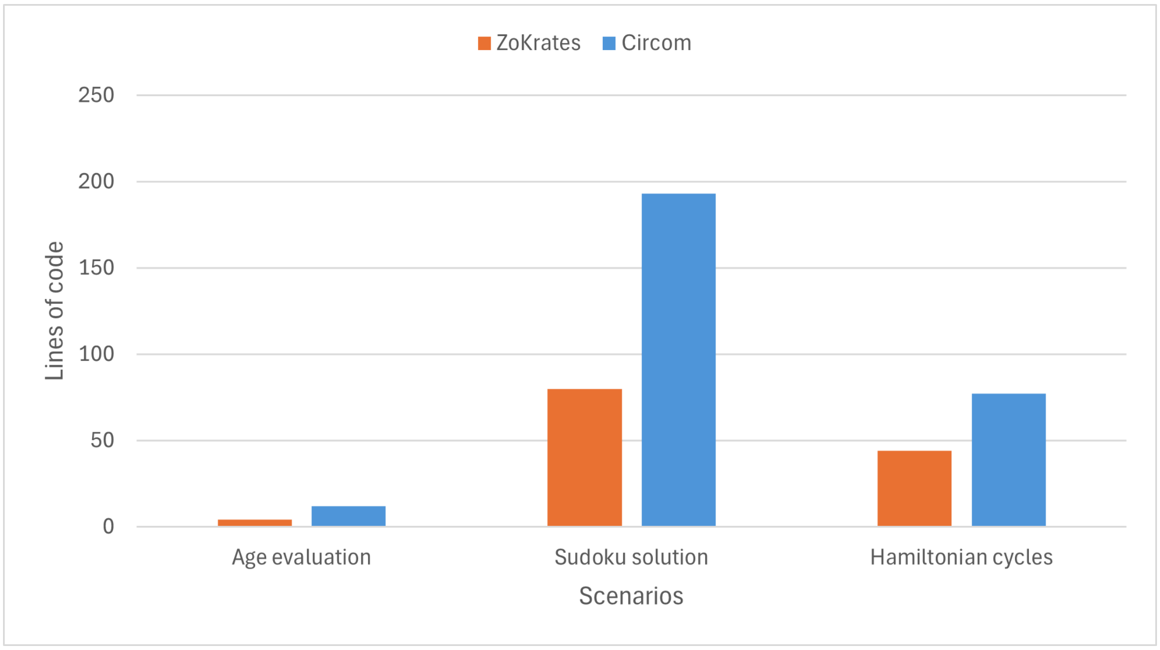

Figure 8: Effective lines of code needed to code the scenarios using the two libraries

an easier adoption of the tool. In the light of these considerations, the choice of one library instead of the other has to be weighted to several factors, such as developer comfort and skills, available off-chain storage, problem complexity.

7. Conclusions

In this paper we propose a comparative study to benchmark the performances of Circom and ZoKrates, two among the major libraries used to generate zkSNARK proofs in blockchain related applications and to verify them on-chain. The comparison was performed considering several scenarios of increasing complexity, in which a prover holds a secret witness and aims to prove the knowledge about such witness without disclosing it to the verifier. We have evaluated the entire workflow, from the proof generation to its verification. The proofs are verified by a smart contract generated by the libraries. The witnesses to prove the knowledge of were: *i)* to be older than a certain age threshold, *ii)* a valid solution for a Sudoku grid, *iii)* a valid hamiltonian cycle for a graph. We implemented several instances for each of these scenarios, increasing the input size, and consequently the computational charge, at each stage. Finally, we evaluated the performances of the two libraries against the mentioned scenario implementations, collecting measurements about several parameters, in particular the verifier smart contract gas costs and the storage costs at Layer 2.

Our findings highlight that the proving workflows produced by Circom appears cheaper in both Layer 2 storage occupation and smart contract costs on every tested scenario. However, even if Circom appears more performing than ZoKrates on the evaluated scenarios, the coding experience provided by the ZoKrates language was overall more fluid and produced more readable and compact code, as showed for example by our analysis of the effective lines of code needed to code the scenarios.

As future works, we plan to enrich the study defining more scenarios involving blockchain related problem, such as the transaction anonymization or the privacy preserving proof of ownership, or scenarios in which features like witness succintness is more crucial. We also plan to evaluate more parameters, such as the time to execute the various workflow steps (proof generation, proof verification, trusted setup…) and the scalability in terms of gas with respect of executing the same statements on a classic Layer 1 Ethereum smart contract. Finally, we will consider to include more libraries in the comparison.

Acknowledgements

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Additionally, it was partially funded by Ministero dell'Università e della Ricerca (MUR), issue D.M. 352/2022 “Borse di Dottorato” - Dottorato di Ricerca di Interesse Nazionale in “Blockchain & Distributed Ledger Technology”, under the National Recovery and Resilience Plan (NRRP).

References

[1] H. Mayer, zk-snark explained: Basic principles, URL https://blog. coinfabrik. com/wp-content/uploads/2017/03/zkSNARK-explained_basic_principles. pdf (2016).

[2] M. Petkus, Why and how zk-snark works, CoRR abs/1906.07221 (2019). URL: http://arxiv.org/abs/1906.07221. arXiv:1906.07221.

[3] D. Di Francesco Maesa, A. Lisi, P. Mori, L. Ricci, G. Boschi, Self sovereign and blockchain based access control: Supporting attributes privacy with zero knowledge, J. Netw. Comput. Appl. 212 (2023) 103577. doi:10.1016/j.jnca.2022.103577.

[4] F. Vincent C. Hu, David, K. Rick, S. Adam, M. Sandlin, K. Robert, S. Karen, Guide to attribute based access control (ABAC) definition and considerations, 2014.

[5] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, IACR Cryptol. ePrint Arch. (2016) 555. URL: http://eprint.iacr.org/2016/555.

[6] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, E. Dutkiewicz, Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities, IEEE Access 7 (2019) 85727–85745. doi:10.1109/ACCESS.2019.2925010.

[7] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Technical Report, 2009.

[8] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) 1–32.

[9] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof systems, SIAM Journal on Computing 18 (1989) 186–208. doi:https://doi.org/10.1137/0218012.

[10] J. Quisquater, M. Quisquater, M. Quisquater, M. Quisquater, L. C. Guillou, M. A. Guillou, G. Guillou, A. Guillou, G. Guillou, S. Guillou, T. A. Berson, How to explain zero-knowledge protocols to your children, in: G. Brassard (Ed.), Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, volume 435 of *Lecture Notes in Computer Science*, Springer, 1989, pp. 628–631. doi:10.1007/0-387-34805-0_60.

[11] M. Bellare, O. Goldreich, On defining proofs of knowledge, in: Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings, volume 740 of *Lecture Notes in Computer Science*, Springer, 1992, pp. 390–420. doi:10.1007/3-540-48071-4_28.

[12] T. Chen, H. Lu, T. Kunpittaya, A. Luo, A review of zk-snarks, 2023. arXiv:2202.06877.

[13] Zcash and zksnarks, "https://z.cash/technology/zksnarks/" [Online, accessed on 23 April 2024], 2016.

[14] Zcash - trusted setup ceremony, "https://z.cash/technology/paramgen/" [Online, accessed on 23 April 2024], 2016.

[15] S. Simunic, D. Bernaca, K. Lenac, Verifiable computing applications in blockchain, IEEE Access 9 (2021) 156729–156745. doi:10.1109/ACCESS.2021.3129314.

[16] J. Partala, T. H. Nguyen, S. Pirttikangas, Non-interactive zero-knowledge for blockchain: A survey, IEEE Access 8 (2020) 227945–227961. doi:10.1109/ACCESS.2020.3046025.

[17] A. E. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: IEEE Symposium on Security and Pri-

vacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, IEEE Computer Society, 2016, pp. 839–858. doi:10.1109/SP.2016.55.

[18] L. T. Thibault, T. Sarry, A. S. Hafid, Blockchain scaling using rollups: A comprehensive survey, IEEE Access 10 (2022) 93039–93054. doi:10.1109/ACCESS.2022.3200051.

[19] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, M. Orrù, zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks, IACR Cryptol. ePrint Arch. (2023) 1503. URL: https://eprint.iacr.org/2023/1503.

[20] K. Baghery, A. Mertens, M. Sedaghat, Benchmarking the setup of updatable zk-snarks, IACR Cryptol. ePrint Arch. (2023) 1161. URL: https://eprint.iacr.org/2023/1161.

[21] C. Steidtmann, S. Gollapudi, Benchmarking zk-circuits in circom, IACR Cryptol. ePrint Arch. (2023) 681. URL: https://eprint.iacr.org/2023/681.

[22] Zk-snark hash benchmark, https://github.com/colinnielsen/SNARK-hash-benchmark/, 2022.

[23] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, J. B. Melé, Circom: A circuit description language for building zero-knowledge applications, IEEE Trans. Dependable Secur. Comput. 20 (2023) 4733–4751. doi:10.1109/TDSC.2022.3232813.

[24] V. Nikolaenko, S. Ragsdale, J. Bonneau, D. Boneh, Powers-of-tau to the people: Decentralizing setup ceremonies, in: International Conference on Applied Cryptography and Network Security, Springer, 2024, pp. 105–134.

[25] J. Eberhardt, S. Tai, Zokrates - scalable privacy-preserving off-chain computations, in: IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018, IEEE, 2018, pp. 1084–1091. doi:10.1109/Cybermatics_2018.2018.00199.

[26] J. Groth, On the size of pairing-based non-interactive arguments, IACR Cryptol. ePrint Arch. (2016) 260. URL: http://eprint.iacr.org/2016/260.

[27] K. George, The mathematical mechanics behind the groth16 zero-knowledge proving protocol (2022).

[28] B. Felgenhauer, F. Jarvis, Mathematics of sudoku i, Mathematical Spectrum 39 (2006) 15–22.

[29] M. Claverol, A. G. Olaverri, D. Garijo, C. Seara, J. Tejel, On hamiltonian alternating cycles and paths, Comput. Geom. 68 (2018) 146–166. doi:10.1016/J.COMGEO.2017.05.009.