Comparative Analysis and Implementation of Zero-Knowledge Proof Libraries for Digital Identity*

Aristidas Lukas Končius[1], Gintarė Košubienė[1], Laura Atmanavičiūtė[1] and Saulius Masteika[1]

[1]Vilnius University, Kaunas Faculty, Muitinės g. 8, 44280 Kaunas, Lithuania

Abstract

Zero-knowledge proofs (ZKPs) enable identity verification without exposing sensitive data, making them essential for decentralized digital identity. This study compares zk-SNARKs and zk-STARKs regarding security, scalability, and efficiency to determine the most suitable approach. The results show zk-SNARKs to be more efficient because of their smaller proof sizes and constant time verification, making them well suited for the efficiency demands of identity verification. Experimental comparison of zk-SNARK libraries showed ZoKrates to be the most practical, offering a balance between usability and performance, mainly due to its JavaScript compatibility. A digital identity age verification app was implemented using ZoKrates to verify these outcomes and demonstrate its viability. This research sets the stage for future development in proof generation efficiency and setup mechanisms.

Keywords

Zero-knowledge proof, digital identity, zk-STARK, zk-SNARK, ZoKrates

1. Introduction

As digital interactions become an essential part of modern life, digital identity plays a crucial role in providing access to online services, financial platforms, and government applications. Digital identity management systems are categorized into two main models: centralized and self-sovereign identity (SSI) models [1]. The centralized model, the most widely adopted, relies on intermediaries storing user data in a single repository, making it vulnerable to cyberattacks, privacy breaches, and single points of failure [2]. While regulations like GDPR aim to improve transparency, they fail to fully mitigate data misuse risks [1].

To address these limitations, SSI shifts identity control to individuals, leveraging blockchain technology for security and decentralization. Unlike centralized databases, SSI allows users to own, store, and selectively share their identity attributes without intermediaries [1]. However, blockchain's inherent transparency introduces privacy challenges, which can be effectively addressed using zero-knowledge proofs (ZKP).

1.1. Why is ZKP Important for Decentralized Digital Identity

Zero-knowledge proofs solve several critical problems in blockchain based identity management systems by enabling private, trustless and secure identity verification without revealing sensitive data [1]. By allowing users to prove their identity attributes without disclosing personal details, ZKPs address key challenges in decentralized identity systems:

- Privacy risk on blockchain
- Reliance on central authorities
- Identity fraud and Sybil attacks
- Efficiency in identity verification

By addressing these challenges, ZKP proves its importance for decentralized SSI identity management.

*IVUS 2025: Information Society and University Studies 2025, May 15, Kaunas, Lithuania

aristidas.koncius@knf.stud.vu.lt (A. L. Končius); gintare.kosubiene@knf.vu.lt (G. Košubienė); laura.atmanaviciute@knf.vu.lt (L. Atmanavičiūtė); saulius.masteika@knf.vu.lt (S. Masteika)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-06-12

CEUR-WS.org/Vol-4213/paper10.pdf

1.2. Problem

Zero-knowledge proofs (ZKP) address key challenges of decentralized identity management systems by enabling verification without exposing sensitive data, with zk-SNARKs and zk-STARKs being the leading approaches. However, a clear consensus on which approach, as well as which library within the selected approach, is more suitable for specific real-world applications is lacking and practical implementations of these systems remain underexplored in existing research.

1.3. Aim and Objectives

This research aims to determine the most suitable ZKP approach for decentralized digital identity verification by comparing zk-SNARKs and zk-STARKs in terms of security, scalability and efficiency. Based on this comparison, it will then evaluate the available libraries of the selected approach to identify the most practical solution for real-world applications, ensuring a balance between usability, computational performance and practical feasibility. Additionally, the study will demonstrate its implementation in a decentralized identity verification use case.

To achieve this aim, the following objectives have been established:

1. Examine the fundamental principles of of ZKP and its different types.
2. Conduct a detailed comparative analysis of zk-SNARKs and zk-STARKs, evaluating their security, scalability and computational efficiency.
3. Determine the most suitable ZKP approach for decentralized digital identity based on the comparative analysis.
4. Experimentally compare selected ZKP approach libraries and determine which is the best one based on usability, computational performance and practical feasibility.
5. Implement a decentralized identity verification use case using the selected ZKP approach and its most practical library.

2. Understanding Zero-Knowledge Proofs

2.1. Essential Characteristics of Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs) are cryptographic protocols that allow a prover to convince a verifier that a statement is true without revealing any additional information [3]. Introduced by Goldwasser, Micali, and Rackoff in the 1980s, ZKPs are now a fundamental component of privacy-preserving cryptography [4].

A valid ZKP must satisfy three properties [5]: [5]:

- Zero-knowledge: it is impossible for a verifier to extract any information except whether the statement is true or not.
- Completeness: a verifier can be convinced that the statement provided by the prover is true.
- Soundness: a prover can't convince the verifier of a statement that is not true with a sufficiently high probability.

2.2. Interactive vs. Non-Interactive Zero-Knowledge Proofs

Zero-knowledge proofs can be classified into two different types: interactive and non-interactive proofs. They are based on the nature of communication between the prover and the verifier.

Interactive ZKPs require multiple rounds of interaction, where the verifier challenges the prover to confirm their knowledge [5]. While effective, this approach is inefficient for large-scale applications since each verifier must individually interact with the prover.

Non-interactive ZKPs (NIZKPs) eliminate this back-and-forth communication, allowing a single proof to be verified at any time by anyone [5]. This is achieved using the Fiat-Shamir heuristic, which replaces verifier challenges with cryptographic hash functions [6].

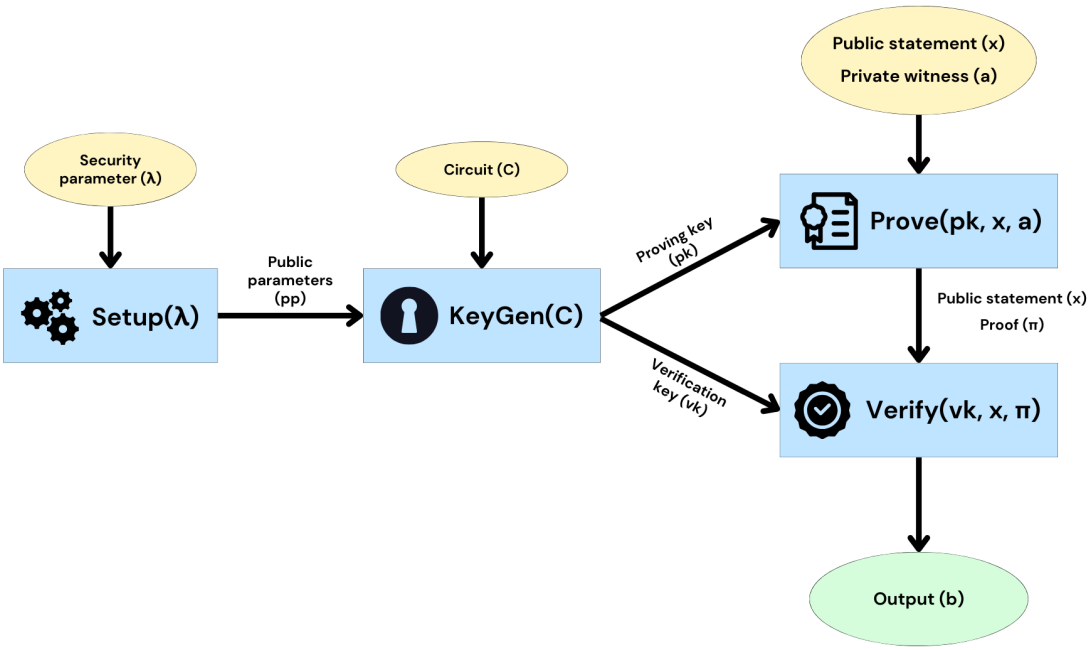

Figure 1: zk-SNARKs algorithms scheme

2.3. Deep Dive into zk-SNARKs

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) are a cryptographic proof system that allows a prover to convince a verifier of the validity of a computation without revealing any underlying data [1]. Zk-SNARKs were first introduced by Ben-Sasson, Chiesa, Genkin, Tromer and Virza in 2013 [7].

Four main algorithms of the zk-SNARK system:

1. Setup algorithm (Setup$(1^\lambda) \rightarrow pp$). Generates public parameters (pp) using a security parameter λ, defining the common reference string (CRS) for proof construction [1].
2. Key generation algorithm (KeyGen(C) $\rightarrow$ (pk, vk)). Generates a proving key (pk) and a verification key (vk) based on a circuit (C) [1].
3. Proof generation algorithm (Prove(pk, x, a) $\rightarrow \pi$). Generates a non-interactive proof (π) based on a proving key (pk), public statement (x) and a private witness (a) [1].
4. Proof verification algorithm (Verify(vk, x, π) $\rightarrow$ b). Determines proof validity using the verification key (vk), public statement (x) and proof (π).

Figure 1 illustrates a visual representation of the explained zk-SNARK algorithms, enhancing the comprehension of their relationships.

zk-SNARKs achieve constant-time verification (O(1)) [8], ensuring efficiency regardless of computation size. Their small proof sizes make them highly practical, as the number of elements needed for verification remains minimal. Proof sizes for the Groth16 scheme vary depending on the elliptic curve used:

- Groth16 (BN254): 192 bytes. Groth16 proofs consist of 3 group elements [8] and each element is 64 bytes size using BN254 pairing-friendly elliptic curve.
- Groth16 (BLS12-381): 288 bytes. Groth16 proofs consist of 3 group elements [8] and each element is 96 bytes size using BLS12-381 pairing-friendly elliptic curve.

Modern CPUs with native pairing implementations enable Groth16 verification in under 0.001 seconds, making zk-SNARKs highly efficient for real-world applications [9].

2.4. Deep Dive into zk-STARKs

Zero-Knowledge Scalable Transparent Arguments of Knowledge (zk-STARKs) were introduced by Eli Ben-Sasson, Iddo Bentov, Yinon Horesh and Michael Riabzev [10] to address zk-SNARK limitations, particularly the need for a trusted setup [10]. Unlike zk-SNARKs, zk-STARKs replace elliptic curve pairings with hash-based commitments and error-correcting codes, enhancing transparency, scalability, and quantum resistance [11]. They employ the Reed-Solomon interactive oracle proof (FRI) protocol for low-degree testing and the Fiat-Shamir heuristic to achieve non-interactivity via cryptographic hash functions [10]. zk-STARKs operate through five main algorithms:

1. Setup. Defines security parameters, polynomial constraints, and a cryptographic hash function for randomness [1].
2. Encoding: Converts inputs into finite field elements, enabling mathematical representation of computations [1].
3. Trace computation. Represents computation as structured polynomials, using Reed-Solomon codes and Fast Fourier Transform (FFT) for efficiency [1].
4. Proof generation. Constructs a compact proof with secret polynomial randomization and Reed-Solomon encoding for integrity [1].
5. Proof verification. Checks polynomial constraints and cryptographic commitments without re-executing the computation.

zk-STARK proofs typically range from 40 KB to 1 MB, depending on computation complexity, compared to 192–288 bytes for zk-SNARKs [10] [12]. Their logarithmic-time verification O(log T(n)) ensures efficiency even as computation scales, with empirical benchmarks showing sub-0.1s verification for computations up to 34 billion gates [10].

2.5. Comparing zk-SNARKs and zk-STARKs

While both zn-SNARKs and zk-STARKs enable privacy-preserving computation and efficient verification, they differ in key aspects such as proof size, verification time, security assumptions, setup requirements, main methods used and scalability. A comparison of these systems is provided in Table 1.

Table 1
Comparison of zk-SNARKs and zk-STARKs

Feature	zk-SNARKs	zk-STARKs
Proof size	Small (192–288 bytes using Groth16)	Larger (40 KB – 1 MB, depending on computation size)
Verification time	Very fast (O(1))	Fast but slightly slower (O(log T(n)))
Trusted setup	Yes	No
Mathematical foundation and security	Based on elliptic curve cryptography and bilinear pairings	Uses hash-based commitments, Reed-Solomon codes and collision-resistant hash functions
Quantum resistance	No	Yes
Scalability	Good for small computations	Optimized for large computations

One of the main differences between zk-SNARKs and zk-STARKs is the requirement for a trusted setup in zk-SNARKs. zk-SNARKs rely on a trusted setup ceremony to generate the CRS necessary to verify proof. If this setup is compromised, the entire system's security is at risk. In contrast, zkSTARKs eliminate the need for a trusted setup, ensuring transparency and verifiability without reliance on a predefined setup phase. This makes zk-STARKs more suitable for applications requiring high transparency and trust minimization.

Another important feature when comparing these two systems is their proof size, verification time and overall scalability. zk-SNARKs proofs are more compact, ranging from 192 to 288 bytes when

using the Groth16 algorithm. In contrast, zk-STARKs proofs are larger, ranging from 40KB to 1MB, depending on the computation's complexity. Proof verification time is also faster for zk-SNARKs, it achieves constant time verification O(1), which is highly efficient for smaller computations but does not scale well for substantial tasks. zk-STARKs have logarithmic time verification O(log T(n)), which slightly increases verification complexity as computation grows but remains scalable for large systems. This makes zk-SNARKs better for applications requiring minimal proof sizes and fast verification, while zk-STARKs excel in large-scale applications where scalability is a priority.

zk-SNARKs rely on elliptic curve cryptography and bilinear pairings, which provide strong security guarantees under existing hardness assumptions. However, this reliance on elliptic curves makes zk-SNARKs vulnerable to potential quantum attacks. zk-STARKs use hash-based commitments and Reed-Solomon codes to provide post-quantum security. Their reliance on collision-resistant hash functions instead of traditional number-theoretic assumptions provides robustness against quantum computing attacks, making zk-STARKs a more sustainable and future-proof solution.

Both zk-SNARKs and zk-STARKs offer powerful cryptographic techniques for preserving privacy, but their use cases differ. zk-SNARKs are more efficient in proof size and verification time, making them preferable for applications where efficiency is key. However, zk-STARKs provide higher transparency, quantum resistance and superior scalability, making them an attractive choice for large-scale applications and long-term security.

2.6. Choosing Zero-Knowledge Proof for Digital Identity

zk-SNARKs are the preferred choice for digital identity applications due to their small proof size (192–288 bytes), ensuring efficient storage, transmission, and validation. In contrast, zk-STARK proofs (40 KB–1 MB) are significantly larger, making them less practical for identity verification, where compactness is crucial.

Verification speed is another key factor. zk-SNARKs enable instant proof validation with O(1) complexity, whereas zk-STARKs follow O(log T(n)) verification, making them less efficient for tasks such as age verification, diploma authentication, and document validity checks, which involve simple computations.

While zk-STARKs excel in large-scale computations and provide post-quantum security, these benefits are less relevant for identity verification, where low-latency proof validation is the priority. Although zk-SNARKs require a trusted setup, this risk is mitigated using multi-party computation (MPC) for secure CRS generation.

Considering proof size, verification efficiency, and practicality, zk-SNARKs offer the optimal balance for decentralized digital identity systems.

3. Choosing library for building with zk-SNARKs

To evaluate the performance, complexity, and usability of various zk-SNARK libraries implementing Groth16, an age verification use case was implemented across different libraries. This benchmark simulates a real-world digital identity use case, where users prove they meet an age requirement without revealing their birthdate.

Each library underwent five test runs, with proof generation time, verification time, and total execution time recorded. The average values were calculated to minimize runtime variation. To assess implementation complexity, the number of lines of core implementation code (excluding comments and whitespace) was measured. Since programming language affects integration and usability, the primary language of each library was also documented. Additionally, ease of use was evaluated based on documentation quality, learning curve, and required cryptographic knowledge. The results of the conducted experiment are presented in Table 2.

Gnark demonstrated the best raw performance, achieving 6.3 ms proof generation and 1.0 ms verification, making it ideal for low-latency applications. However, its Go-based implementation

Table 2
Comparison of zk-SNARK Libraries Using Groth16

Library	Avg. Proof Generation Time (ms)	Avg. Proof Verification Time (ms)	Avg. Total Execution Time (ms)	Lines of Code	Programming Language	Ease of Use
ZoKrates	728.2	60.8	3260.4	16	JavaScript	Beginner friendly
SnarkJS	1452.0	681.6	7044.4	42	JavaScript	Intermediate-advanced
Gnark	6.3	1.0	78.8	35	Go	Intermediate
Bellman	179.2	14.2	1694.4	86	Rust	Very complex*
Arkworks	89.9	16.3	329.7	47	Rust	Advanced

*Uses finite field arithmetic.

complicates web integration, unlike JavaScript libraries such as ZoKrates and SnarkJS, which offer better compatibility.

SnarkJS, despite being JavaScript-based, had the highest execution times (1452.0 ms proof generation, 681.6 ms verification), making it the least efficient option. Bellman and Arkworks, both Rust-based, balance efficiency and flexibility, but require deeper cryptographic knowledge. Bellman is the most complex, demanding 86 lines of code and an understanding of finite field arithmetic, while Arkworks, at 47 lines, is slightly more accessible.

ZoKrates emerged as the most practical choice for digital identity applications, offering the best balance between usability, efficiency, and integration feasibility. While not the fastest, it maintains sufficient performance with a verification time of 60.8 ms, negligible for identity verification tasks. It requires only 16 lines of code, making it the easiest to implement, especially for developers new to zk-SNARKs. Additionally, as a JavaScript library, it integrates seamlessly into web applications without requiring WebAssembly (WASM) compilation, unlike Go or Rust libraries.

4. Implementing zk-SNARKs Using ZoKrates

ZoKrates is a high-level toolbox for implementing zk-SNARKs, allowing developers to generate, deploy, and verify proofs efficiently [13]. Before using ZoKrates in JavaScript, the environment must be set up by installing node and zokrates-js library. Below is the step-by-step implementation of zk-SNARK-based age verification using ZoKrates.

4.1. initialize()

The ZoKrates environment is initialized, providing access to its toolkit. This step represents the setup algorithm (Setup$(1^\lambda) \rightarrow pp$) where public parameters (pp) are generated.

```
const zokratesProvider = await initialize();
```

Listing 1: Initialization

4.2. compile(sourceCode: string)

A ZoKrates program defining an age verification check is compiled into an intermediate circuit representation.

```
const source = `
    def main(private u32 age) -> bool {
        return age >= 18u32;
    }
`;
const artifacts = zokratesProvider.compile(source);
```

Listing 2: ZoKrates program compilation

```
Proof:  {
  scheme: 'g16',
  curve: 'bn128',
  proof: {
    a: [
      '0x27373fd71638d99d7f8b3d124c294a3178e6b287bf8c7f43a25653eda873ea85',
      '0x1dface35b15a8623c414e7f17f1ca5fe2ed1b7555f7a6f9a1751b9e8ad361bbf'
    ],
    b: [ [Array], [Array] ],
    c: [
      '0x25e2c2b096b8adea0ae83d0b0817e93e64e0eb8ce97b4ac0fb60cd20adc8cc9b',
      '0x27780db7b5b72096353a4084befff58912eafe0a09442b8125cc007f45288807'
    ]
  },
  inputs: [
    '0x0000000000000000000000000000000000000000000000000000000000000001'
  ]
}
```


Figure 2: Proof

4.3. setup(program: Uint8Array)

Setup function requires compiled ZoKrates program in Uint8Array and generates a proving key and a verification key. It represents key generation algorithm (KeyGen(C) $\rightarrow$ (pk, vk)) which generates a proving key (pk) and verification key (vk) based on a circuit (C). Compiled ZoKrates program represents circuit in this function.

```
const keypair = zokratesProvider.setup(artifacts.program);
```

Listing 3: Keypair generation

4.4. computeWitness(artifacts: Artifacts, inputs: string[])

A witness is generated based on input data, capturing execution traces necessary for proof generation.

```
const { witness, output } = zokratesProvider.computeWitness(artifacts, ["25"]);
```

Listing 4: Witness generation

4.5. generateProof(program: Uint8Array, witness: Uint8Array, provingKey: Uint8Array)

A zk-SNARK proof is generated from the witness and proving key. This corresponds to the proof generation algorithm (Prove(pk, x, a) $\rightarrow \pi$).

```
const proof = zokratesProvider.generateProof(artifacts.program, witness, keypair.pk);
```

Listing 5: Proof generation

Figure 2 shows the content of the generated proof object.

4.6. verify(verificationKey: Uint8Array, proof: Proof)

The generated proof is verified against the verification key, determining its validity. It represents the proof verification algorithm (Verify(vk, x, π) $\rightarrow$ b).

```
const isVerified = zokratesProvider.verify(keypair.vk, proof);
```

Listing 6: Proof verification

This was the whole implementation. Figure 3 illustrates the step-by-step workflow of generating and verifying a zk-SNARK proof using ZoKrates.

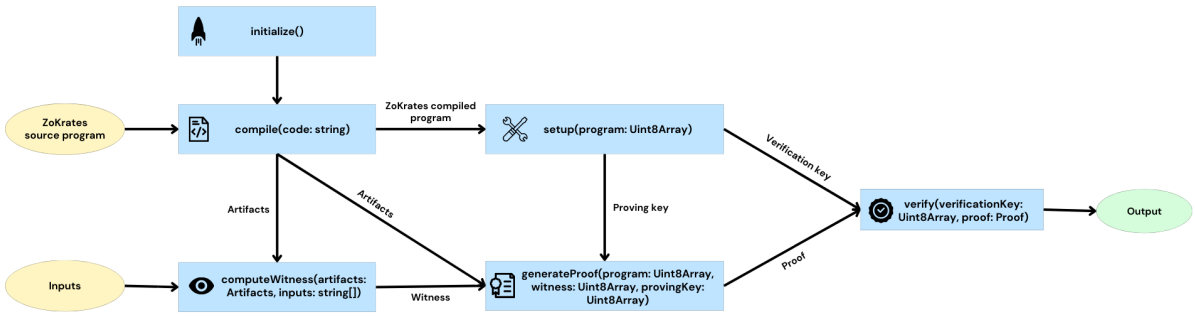

Figure 3: ZoKrates scheme

5. Discussion of Findings in Relation to Existing Research

Our study builds upon and extends the findings of [1] and [9] by providing a comparative analysis of zk-SNARKs and zk-STARKs together with experimental analysis of zk-SNARKs libraries for digital identity verification.

[1] compared zk-SNARKs and zk-STARKs, emphasizing zk-SNARKs efficiency and widespread adoption while acknowledging zk-STARKs security and transparency advantages. [1] also highlighted the importance of ZKP in decentralized identity for privacy and fraud prevention. Our research confirms these insights but moves beyond theoretical comparisons by proving zk-SNARKs' superiority over zk-STARKs in real-world digital identity use case.

[9] analyzed zk-SNARK libraries and identified gaps between theory and practice, taking a broad approach without a specific focus on identity verification. Our research narrows this scope testing ZoKrates, SnarkJS, Gnark, Bellman and Arkworks in the context of digital identity. We confirm [9] findings on implementation challenges but go further by identifying ZoKrates as the most practical choice, offering the best balance of usability, integration feasibility, and performance. Experimental results show that ZoKrates requires only 16 lines of code, significantly lower than Bellman (86) and Arkworks (47), making it the easiest to implement. It also achieves an average proof generation time of 728.2 ms and verification time of 60.8 ms which is unnoticeable in real-world digital identity verification. Unlike [9] generalized evaluation, we validate our conclusions with a real-world implementation, demonstrating that ZoKrates is both efficient and deployable in decentralized identity systems.

By bridging the gap between theoretical comparisons and practical application, our study not only confirms prior research but also provides a definitive recommendation: zk-SNARKs with ZoKrates are the optimal choice for digital identity verification.

6. Conclusion

This study explored the role of zero-knowledge proofs for decentralized digital identity, focusing on a quantitative comparison of zk-SNARKs and zk-STARKs in terms of security, scalability, and efficiency. Through a detailed evaluation, zk-SNARKs proved to be the more viable choice for digital identity

applications, with proof sizes ranging from 192 to 288 bytes using the Groth16 algorithm, compared to zk-STARKs, which produce proofs ranging from 40 KB to 1 MB depending on computation complexity. Additionally, zk-SNARK verification operates in constant time O(1), allowing identity verification in under 0.001 seconds on modern CPUs, while zk-STARKs have a logarithmic verification time O(log T(n)), leading to slower verification for large-scale computations. Given that identity verification typically involves small, discrete computations, zk-SNARKs provide a faster and more storage-efficient solution than zk-STARKs.

Through an experimental comparison of zk-SNARK libraries, ZoKrates emerged as the most practical choice based on usability, computational performance, and integration feasibility for real-world implementations. ZoKrates required only 16 lines of code to implement an age verification proof, significantly less than Bellman (86 lines) and Arkworks (47 lines), making it the simplest library to use. Performance tests showed that ZoKrates achieved an average proof generation time of 728.2 milliseconds and a verification time of 60.8 milliseconds, which is computationally negligible for real-world identity verification. Although Gnark demonstrated the fastest proof generation (6.3 ms) and verification time (1.0 ms), its Go-based environment complicates integration with web applications, unlike ZoKrates, which is JavaScript-compatible and easily deployable in browser-based identity systems.

To validate the feasibility of zk-SNARKs in digital identity management, this study implemented a real-world identity verification use case using ZoKrates. The implementation showed that ZoKrates is able to provide efficient, straightforward, JavaScript compatible proof generation and its verification.

These findings support the potential of zero-knowledge proofs in enhancing privacy and security within decentralized digital identity. Conducted study lays a foundation for future research by identifying zk-SNARKs as best ZKP approach and ZoKrates as the most practical library for implementation. Future work could focus on broader scope of digital identity scenarios such as document authentication, KYC processes or academic credential verification, in this way providing a more comprehensive understanding of practical limitations and strengths. Another important direction is the evaluation of ZKP performance on mobile and edge devices, where computational and storage constraints can significantly impact usability. Lastly, future work could focus on optimizing proof generation time by improving circuit designs, using alternative proving systems or parallel computation strategies.

Acknowledgments

The publication was prepared during the implementation of the project “Implementation of R&D activities, creating APV products by Deverium, UAB" (Project No. 02-020-K-0034). The project is co-financed by the European Union. Findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Deverium, UAB.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4o and Grammarly in order to: Grammar and spelling check. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

[1] L. Zhou, A. Diro, A. Saini, S. Kaisar, P. C. Hiep, Leveraging zero knowledge proofs for blockchain-based identity sharing: A survey of advancements, challenges and opportunities, Journal of Information Security and Applications 80 (2024) 103678. URL: https://linkinghub.elsevier.com/retrieve/pii/S2214212623002624. doi:10.1016/j.jisa.2023.103678.

[2] J. Mirkovic, P. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, SIGCOMM Comput. Commun. Rev. 34 (2004) 39–53. URL: https://dl.acm.org/doi/10.1145/997150.997156. doi:10.1145/997150.997156.

[3] R. P. Barbosa, F. Smarandache, Neutrosophic One-Round Zero-Knowledge Proof, Plithogenic Log. Comp. 2 (2024) 49–54. URL: https://sciencesforce.com/index.php/plc/article/view/363. doi:10.61356/j.plc.2024.2363.

[4] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems, in: Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85, ACM Press, Providence, Rhode Island, United States, 1985, pp. 291–304. URL: http://portal.acm.org/citation.cfm?doid=22145.22178. doi:10.1145/22145.22178.

[5] A. Berentsen, J. Lenzi, R. Nyffenegger, An Introduction to Zero-Knowledge Proofs in Blockchains and Economics, r 105 (2023). URL: https://www.stlouisfed.org/publications/review/2023/05/12/an-introduction-to-zero-knowledge-proofs-in-blockchains-and-economics. doi:10.20955/r.105.280-94.

[6] A. Fiat, A. Shamir, How To Prove Yourself: Practical Solutions to Identification and Signature Problems, in: A. M. Odlyzko (Ed.), Advances in Cryptology — CRYPTO' 86, volume 263, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 186–194. URL: http://link.springer.com/10.1007/3-540-47721-7_12. doi:10.1007/3-540-47721-7_12, series Title: Lecture Notes in Computer Science.

[7] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, M. Virza, SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge, in: R. Canetti, J. A. Garay (Eds.), Advances in Cryptology – CRYPTO 2013, volume 8043, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 90–108. URL: http://link.springer.com/10.1007/978-3-642-40084-1_6. doi:10.1007/978-3-642-40084-1_6, series Title: Lecture Notes in Computer Science.

[8] J. Groth, On the Size of Pairing-Based Non-interactive Arguments, in: M. Fischlin, J.-S. Coron (Eds.), Advances in Cryptology – EUROCRYPT 2016, volume 9666, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 305–326. URL: http://link.springer.com/10.1007/978-3-662-49896-5_11. doi:10.1007/978-3-662-49896-5_11, series Title: Lecture Notes in Computer Science.

[9] J. Liang, D. Hu, P. Wu, Y. Yang, Q. Shen, Z. Wu, SoK: Understanding zk-SNARKs: The Gap Between Research and Practice, 2025. URL: https://arxiv.org/abs/2502.02387. doi:10.48550/ARXIV.2502.02387, version Number: 2.

[10] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, Scalable Zero Knowledge with No Trusted Setup, in: A. Boldyreva, D. Micciancio (Eds.), Advances in Cryptology – CRYPTO 2019, volume 11694, Springer International Publishing, Cham, 2019, pp. 701–732. URL: https://link.springer.com/10.1007/978-3-030-26954-8_23. doi:10.1007/978-3-030-26954-8_23, series Title: Lecture Notes in Computer Science.

[11] A. Berentsen, J. Lenzi, R. Nyffenegger, A Walk-through of a Simple Zk-STARK Proof, SSRN Journal (2022). URL: https://www.ssrn.com/abstract=4308637. doi:10.2139/ssrn.4308637.

[12] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, Scalable, transparent, and post-quantum secure computational integrity, Cryptology ePrint Archive, Paper 2018/046, 2018. URL: https://eprint.iacr.org/2018/046.

[13] ZoKrates Team, ZoKrates: A Toolbox for zk-SNARKs, 2025. URL: https://zokrates.github.io/introduction.html, accessed: 2025-02-26.