On Using zk-SNARKs and zk-STARKs in Blockchain-Based Identity Management

Andreea-Elena Panait[1,2](✉) and Ruxandra F. Olimid[1,3,4](✉)

[1] Department of Computer Science, University of Bucharest, Bucharest, Romania
andreea-elena.panait@drd.unibuc.ro, ruxandra.olimid@fmi.unibuc.ro
[2] certSIGN, Bucharest, Romania
[3] Department of Information Security and Communication Technology, NTNU - Norwegian University of Science and Technology, Trondheim, Norway
[4] The Research Institute of the University of Bucharest (ICUB), Bucharest, Romania

Abstract. One possible applicability of blockchain technology is in identity management. Especially for public blockchains, the need to reduce (ideally to zero) the exposure of sensitive identification data is clear. Under these settings, zero-knowledge proofs, in particular in the advanced forms of *Zero-Knowledge Succinct Non-Interactive ARguments of Knowledge (zk-SNARK)* and *Zero-Knowledge Scalable Transparent ARguments of Knowledge (zk-STARK)*, can be used as a potential privacy-preserving technique. The current work looks at the existing libraries that implement zk-SNARKs and zk-STARKs and exemplifies and discusses the use of zk-SNARKs in blockchain-based identity management solutions.

Keywords: Identity management · Blockchain · zk-SNARK · zk-STARK

1 Introduction

There is a high demand for secure, efficient, and interoperable digital identification nowadays. This is a direct consequence of the increasing number of parties (e.g., users, devices, services) that need to access and operate in the digital environment. Identification is a prerequisite and a first-step for functionalities such as access-control, permissions, confidential communication, etc.

Blockchain technology is a candidate for enhancing identity management by introducing decentralization and other advantages (e.g., *self-sovereignty*, which enables the user to own and control his identity). Nevertheless, for public blockchains, the transaction details might contain sensitive data, and therefore it is important to minimize the exposure of such data within certain use cases. Consequently, there has been a growing interest in using privacy-enhancing techniques for this type of blockchain [50]. Despite the added complexity, zero-knowledge proofs, particularly *Zero-Knowledge Succinct Non-Interac-tive ARguments of Knowledge (zk-SNARKs)* [8] and *Zero-Knowledge Scalable Transparent ARguments of Knowledge (zk-STARKs)* [4] seem promising.

We provide a comparison between zk-SNARKs and zk-STARKs and an overview of the existent libraries that implement them. We refer to how zk-SNARKs might be used for identity management on the blockchain and present use-cases. We give practical examples of zk-SNARK programs for verification of identity attributes compliance, for which we provide measurements in terms of generation time and verification costs.

The remaining of the paper is organized as follows. Section 2 presents the related work. Section 3 gives the necessary background. Section 4 presents available zk-SNARK and zk-STARK libraries. Section 5 exemplifies the applicability of zk-SNARKs for blockchain identity management. Section 6 discusses the measurements results, security aspects, and limitations. Section 7 concludes.

2 Related Work

The concept of *zero-knowledge proof* (zk-proof) was introduced in [27] by Goldwasser et al. Later, Blum et al. showed that non-interactive zk-proofs can exist in the computational settings under the assumption of a Common Reference String (CRS) [9]. Since then, several positive and negative results have been given for zk-proofs in different models. Succinct zk-proofs were first presented in [34], and succinct non-interactive zero-knowledge (NIZK) has been discussed in [41]. SNARKs were first built in [8], together with some applicability (e.g., delegation of computation, two-party secure computation), under the assumption that extractable collision-resistant hash functions exist. In [28], Groth presents the NIZK arguments in sub-linear size and give a reduction to a constant number of group elements for a large CRS. Succinct arguments of NP-statements that are fast to construct and verify, using Quadratic Span Programs instead of Probabilistically Checkable Proofs (PCPs) were introduced in [25].

Various zero-knowledge proving systems were proposed during the years. From these, we mention the Pinocchio system [49], SNARKs for C [5], Geppetto computation [18], and NIZK for a von Neumann Architecture [7]. Scalable zero-knowledge was introduced in [6] and later used, for example, in the Coda protocol [40]. Coda uses the recursive composition of zk-SNARKs to obtain a succinct blockchain, removing the blockchain scalability issue. A pairing-based (pre-processing) SNARK for arithmetic circuit satisfiability, which is an NP-complete language, was presented in [29]. Based on this, various implementations were given [10,30,56]. *Bulletproofs*, a more recent type of constraint zk-proofs (i.e., the statement cannot be general), was introduced in [13]. zk-STARKs were proposed in [4]. Valuable overviews on SNARKs, STARKs, and bulletproofs are given in [13,50]. Recent proving systems include Sonic [39], Halo [11] and Libra [57]. Distributed zk-proof generation was proposed in [56]. We recall a special type of zk-proofs that was introduced in [24], somehow connected to our field of interest, the *zero-knowledge proof-of-identity* which proves a party's identity by demonstrating the knowledge of the private key that corresponds to the party's public key. Much subsequent work followed, which we deliberately omit here.

The usability of zk-proofs (in particular zk-SNARKs and zk-STARKs) in blockchain-based solutions has been discussed in several scenarios [50]. To exemplify, in [54], the authors presented an interoperable healthcare blockchain-based system that uses zk-proofs to authenticate the beneficiaries, Zcash uses zk-SNARKs to create shielded transactions [59], and zero-knowledge proof-of-identity has been proposed to overcome Sybil attacks [15]. A proposal of an identity management system that claims to preserve privacy in the blockchain by the usage of zk-SNARKs was described in [38]. Currently, not much literature exists on zk-SNARKs and zk-STARKs usage for identity management on the blockchain. But a considerable number of proposals exist for identity management on blockchain in general. A brief survey of current work and existing solutions can be found in [48]. However, despite the apparently large number of such solutions, their maturity remains still questionable [47].

3 Background

3.1 Blockchain-Based Identity Management

A blockchain stores data in a *descentralized* and *distributed* manner, by using nodes that agree on the stored data using a *consensus* protocol. The data is stored in *blocks*, which are collections of *transactions*. A nice overview on blockchain is given in [58]. From the various blockchain implementations, we mention *Ethereum* [22], which we will refer to in the paper because of its ability to implement *smart contracts*. Smart contracts allow execution via a function-based interface stored in the blockchain and hence might be used for identity and attributes verification of the parties. Every operation performed in the Ethereum blockchain (e.g., simple transactions, smart contract executions) requires *gas*.

A blockchain-based identity management solution uses the blockchain capabilities to implement identity management functionalities. The identities can be attested either by recognized authorities or by other entities in the blockchain (normally considered of trust and above a certain threshold number). Each entity has an *identifier* and some *attributes*, which can be stored on- or off-chain [19]. A more detailed look over the identity management on blockchain is given in [48].

3.2 Zero-Knowledge Protocols

In a zero-knowledge protocol, a *prover* assures a *verifier* about the validity of a statement, without revealing any information other than its validity. It is guaranteed that a malicious prover cannot fool the verifier to accept a false statement (*soundness*). A more relaxed notion, named *zero-knowledge arguments*, computationally bounds the capabilities of the malicious prover to polynomial strategies (*computational soundness*) [12]. Throughout the paper, we will refer to *zero-knowledge proofs-of-knowledge* for which a prover can also demonstrate that he knows a *witness* that satisfies the statement. The zero-knowledge property assures that the proof does not disclose or damage the secrecy of the witness. They exist in both *interactive* (i.e., requires interaction between the prover and the verifier) and *non-interactive* (i.e., does not require interaction between the prover and the verifier) versions, with heuristics (e.g., Fiat-Shamir [24]) that can transform the former in the latest under certain conditions or with some necessary changes. For general statements, non-interactive zero-knowledge protocols are only possible under the assumption of a Common Reference String (CRS) that needs to be known by both the prover and the verifier [13].

A type of non-interactive argument of knowledge is the zk-SNARK [8]. Besides the zero-knowledge property, the zk-SNARKs provide the property of *succintness*, meaning that the proofs are small, and the verification is cheap and does not require expensive processing [14,50,56]. Nevertheless, they come with some drawbacks: the necessity of a trusted setup (they work in the CRS model) and (still quite) a significant overhead for the setup and on the prover side [56].

Another zero-knowledge non-interactive construction is the *scalable* and *transparent* argument of knowledge, zk-STARK [3]. Here transparency means that randomness used by the verifier is publicly available, so the necessity of a trusted setup is eliminated [4]. On the drawbacks, the proof size is considerably larger than for zk-SNARKs [4]. Similar to SNARKs, STARKs can be executed with or without *zero-knowledge* and designed to be interactive or non-interactive [3].

3.3 zk-SNARK and zk-STARK Program Representation

Quadratic Arithmetic Program (QAP)-based zk-SNARKs are used for implementing practical use-cases. The predicate statement is internally codified in terms of an arithmetic circuit and based on this codification, appropriate tools can generate zk-SNARKs, by transforming the circuits into a QAP. QAPs are based on pairings over elliptic curves, used to encode the computational steps. Examples of elliptic curves include: the Barreto-Naehrig (BN) curves [2], Edwards [21], MNT [42], BLS12-381 [61]. The arithmetic circuit used in a specific zk-SNARK corresponds to the finite field that underlines the elliptic curve used, and a circuit wire corresponds to a single elliptic curve field element. After assigning values to all of the circuit's wires (the circuit represents a single computation for specific public and private set of inputs), the next step for constructing a zk-SNARK is to provide a specific set of constraints that attest that computation has been correctly performed. This set of constraints represents the Rank-1 Constraint System (R1CS) and is used for preventing a malicious prover to provide a verifier with an output that has not been created from its inputs [50]. Other representing proof systems are considered to be Bilinear Arithmetic Circuit Satisfiability (BACS), Unitary-Square Constraint Systems (USCS), and Two-input Boolean Circuit Satisfiability (TBCS) [36]. BACS internally reduces to R1CS and TBCS internally reduces to USCS, being more efficient than using R1CS [36]. The tools that allow the proof predicate (i.e., the statement) to be stated in a high-level language (so that is easier to learn and use it) are called *Domain-Specific Languages (DSL)* tools. For zk-STARKs, the predicate should be transformed in an Algebraic Intermediate Representation (AIR) or a Permuted Algebraic Intermediate Representation (PAIR) [4].

4 zk-SNARK and zk-STARK Comparison and Available Development Libraries

In this section, we compare zk-SNARKs and zk-STARKs and give an overview of the main libraries that implement them. As already mentioned in Sect. 2, other types of zk-proofs (e.g., bulletproofs) exist but are outside of the goal of this paper. Nice comparisons between more types of zk-proofs are given in [13,50].

zk-SNARKs are defined in the CRS model, so they require an initial trusted setup phase during which parties gain knowledge on a string (which can be further thought of in terms of secret keys) [50,56]. The security of the zk-SNARKs is based on the security of the trusted setup. Hence, if the trusted setup is compromised, the whole system is damaged. On the contrary, zk-STARKs make use of public randomness (they are *transparent*), thus eliminating the need for a secret pre-shared value [4]. In many scenarios, this is a clear advantage of zk-STARKs over zk-SNARKs because the setup might be a much too strong assumption and transparency is necessary for public, distributed trust [4].

The proof size of the zk-SNARKs is small (they are *succint*), and the verification of such proof is fast [14,50,56]. They gain practicality due to the constant proof size, regardless of the statement to be proved (e.g., Groth et al. give a construction for which the proof consists of three group elements [29]). In comparison, zk-STARKs generate much larger proofs (roughly 1000 times larger [4]). In terms of verification, both systems have fast verification time, with zk-SNARKs slightly outperforming zk-STARKs. zk-SNARKs are traditionally based on strong number-theoretical hardness assumptions that do not hold against a quantum adversary [13]. Recently, post-quantum resistant zk-SNARKs were introduced [16,26,45]. In comparison, zk-STARKs are normally post-quantum secure [4] due to the quantum-resistant cryptographic primitives they base on (e.g., collision-resistant hash functions, which are not known to be broken by quantum computers [3]).

4.1 Development Libraries

Several libraries that implement zk-SNARKs and zk-STARKs have been developed. Table 1 lists some of the existing zk-SNARK libraries together with the language in which they are implemented, their representing proof predicate language, the underlying elliptic curves, SNARK constructions that can be used in the library, and the available DSL tools. Table 2 looks into the zk-SNARK DSL tools and lists them with the associated language and the corresponding back-end

Table 1. zk-SNARKs and zk-STARTKs libraries [46,50]

Library	Language	Representing proof predicate language	Eliptic curves	zk-SNARK/ zk-STARKs Types	DSL Tools
libsnark [36]	C++	R1CS; BACS; USCS; TBCS	BN [2]; Edwards [21]; MNT [42]	BCTV14 [7]; Groth16 [29]; GM17 [30]	ZoKrates [62]; JSnark/xjSnark [1,35]; Snarky [37]
DIZK [52]	Java	R1CS	BN254 [2]	Groth16 [29]	–
Snarkjs [33]	Javascript	R1CS	BN254 [2]	BCTV14 [7]; Groth16 [29]	Circom[32]
Bellman [60]	Rust	R1CS	BLS12-381 [61]	Groth16 [29]	ZoKrates [62]
ZEXE [53]	Rust	R1CS	Edwards [21]; MNT [42], BN [2]	Groth16 [29], GM17 [30]	ZEXE's snark-gadgets [53]
libSTARK [3]	C++	–	–	BN18 [4]	–
genSTARK [31]	Javascript	-	-	BN18 [4]	–

Table 2. DSL tools [46,50]

DSL tool	Language	Back-end zk-SNARK Library
ZoKrates [62]	Rust; C++	libsnark [36]; Bellman [60]
JSnark/xjSnark [1,35]	Java	libsnark [36]
Circom [32]	Javascript	Snarkjs [33]
Snarky [37]	OCaml	libsnark [36]
ZEXE's snark-gadgets [53]	Rust	ZEXE [53]

libraries. Table 1 also lists the available development libraries for zk-STARKs. To the best of our knowledge, there are no DSL tools available for zk-STARKs at the moment. Notice the significantly less number of implementations for zk-STARKs, which we assume to be a natural consequence of a later definition of zk-STARKs than zk-SNARKs and a currently lower practical interest.

5 Blockchain-Based Identity Management Using zk-SNARKs

Currently, zk-SNARKs are more suitable to be used in a blockchain due to their better capabilities (e.g., small and constant proof size). In time, if zk-STARKs become more efficient, they could dominate because they do not use a trusted setup [3]. Further, we looked into the applicability of zk-SNARKs into blockchain-based identity management.

5.1 General Architecture

In [38], Lee et al. gave a blockchain-based identity management scheme that makes use of zk-SNARKs and is compatible with the ZoKrates process [20]. We do not claim their proposal is secure, nor discuss other aspects here (this is out of our scope) but only consider the general architecture depicted in Fig. 1 for further testing. Important security considerations are discussed in Sect. 6.

Step 1. A user Alice asks a Certified Authority (CA) to certify her identity and attributes. For security reasons (to prevent disclosure and changes), the certificate $cert_A$ is issued on a modified version of the data that is both *hidding* and *binding*. For simplicity, we further consider this to be a cryptographically hashed value H_A. Step 1 is executed off-chain and the method by which the authority verifies the validity of the identity attributes is out of our interest.

Step 2. The certificate $cert_A$ for H_A is uploaded in the blockchain. The scope of $cert_A$ is to certify that H_A indeed corresponds to the identity and attributes of Alice. As the certificate is publicly verified, this step can be performed either by the CA (step 2") or directly by Alice (step 2').

Step 3. A Third Party publishes on the blockchain a smart contract to verify some attributes of the users. For this, a one-time setup phase takes place, during which a zk-SNARK proving and a verification key are generated. The Third Party securely transmits the proving key to Alice. Based on the verification key, a Solidity smart contract is generated and deployed on the blockchain to further verify the given proofs against the value H_A stored in the blockchain. Note that the setup is independent of the first steps (up to the usage of a standard hash function), so the user might join the system after the smart contract is deployed.

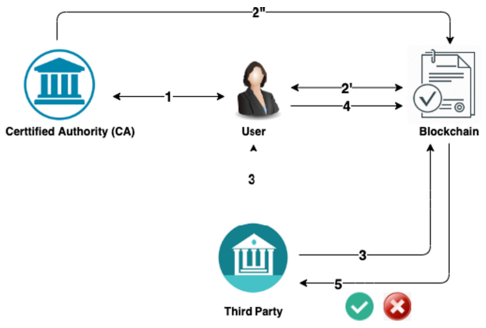

Fig. 1. General architecture of a blockchain-based identity management system that uses zk-SNARKs

Step 4. Alice now wants to prove something about herself (that derives directly from her attributes) to the Third Party without exposing anything else about her attributes. In fact, she does not want to reveal anything else except that her attributes do satisfy a statement (e.g., she is an adult). To do so, she generates a proof that will be verified by the smart contract previously deployed on the blockchain. The proof is generated based on a witness that corresponds to H_A (and to Alice's attributes) and the proving key. Alice sends the generated proof together with the publicly stored value H_A as input to the smart contract.

Step 5. The Third Party checks the result of the smart contract execution by the public address of Alice. If the proof is valid, then Alice proved that she satisfies the statement without revealing anything else about her attributes (under the assumptions of the zero-knowledge property of the proof and the one-way property of H_A). Otherwise, Alice is not able to prove that she satisfies the required statement (this can mean that her attributes do not fulfill the requirements, or that a malicious actor tried to use a fake identity). Note that H_A needs to be available in the public proof so that the validity of $cert_A$ for H_A can be publicly checked (this prevents Alice to use different attributes than the certified ones).

5.2 Use-Cases

We looked into the European Union (EU) regulation on electronic identification and trust services for electronic transactions in the European Single Market (eiDAS) [55]. The eiDAS Security Assertion Markup Language (SAML) Attribute Profile provides the list of attributes included in the eiDAS interoperability framework that supports cross-border identification and authentication processes [55]. The mandatory identity attributes required by the abovementioned regulation are *FamilyName*, *FirstName*, *DateOfBirth*, *PersonIdentifier*, whereas optional attributes are the *BirthName* (either *First Names at Birth* or *Family Name at Birth*), *PlaceOfBirth*, *CurrentAddress*, and *Gender*. The optional attributes may be supplied if available and acceptable by an EU country's national law [55]. We will refer to a simplified example, where we are interested in letting a user prove that he/she is older than a certain age, while not exposing his/her exact age or other information about the age (up to a negligible probability). This is an example that might be useful for online shopping, access to different services, or voting, and it has been previously considered in the literature [38].

5.3 zk-SNARK Implementation

We propose two simplified real-life examples of how zk-SNARKs can be used in identity management. In Sect. 6, we provide measurements of the proposed examples, in terms of proof generation time, amount of Ethereum gas used to deploy the verifier smart contract on a blockchain testnet and the amount of Ethereum gas used for making a verification transaction to the smart contract.

Program 1. zk-SNARK for identity management

```
import "hashes/sha256/512bitPacked.zok" as sha256packed
def main(field pub_year1, field pub_year2, field check_year, field pub_id, private field
year, private field rand) -> (field):
field[2] hash_year = sha256packed([0, pub_id, year, rand])
assert(pub_year1 == hash_year[0])
assert(pub_year2 == hash_year[1])
field rez = if year < check_year then 1 else 0 fi
return rez
```

Program 2. zk-SNARK for identity management

```
import "hashes/sha256/512bitPacked.zok" as sha256packed
def main(field hash_ident1, field hash_ident2, field check_year, field pub_id, private field
year, private field id, private field rand) -> (field):
field[2] result = sha256packed([pub_id, year, id, rand])
assert(hash_ident1 == result[0])
assert(hash_ident2 == result[1])
field rez = if year < check_year then 1 else 0 fi
return rez
```

Tools and Environments. For implementing the zk-SNARK program we used the ZoKrates DSL tool [62]. ZoKrates provides a plugin for Remix IDE tool [51] and enables compiling a proof, computing a witness for the proof, performing the SNARK setup, as well as generating and offering the possibility of exporting the verifier smart contract (that further should be deployed on the blockchain public network). However, ZoKrates is also providing an API, in the *zoktares-js* Javascript library, which can be installed as a Node package [62] and used to perform the above-mentioned steps. Besides those reasons, we choose this tool also because it supports integration with the blockchain network and is easier and straightforward to implement and test such zk-proofs. The ZoKrates zk-SNARK programs were written in Rust language. The default proving scheme for the ZoKrates Remix plugin is the one from [29], and it uses the Bellman library [60] as back-end. To compute the hash value, we used the *sha256packing* function, a component of the standard ZoKrates library [62].

For conducting the tests we used an Intel(R) Core(TM) i7-3632QM CPU @ 2.20 GHz with 8 GB of RAM, Windows 8.1 64-bit Operating System, and Chrome 85 as web browser version. We note that better configurations of the machine used for calculations could improve the results, but to what extent the configuration matters, is not a subject of this research. We also note that by using the Remix IDE tool, the time required for the generation of proof might not be accurate because the calculations are made by calling methods from an interface and not by calling them thought the command line.

zk-SNARK Usage Examples. Similar to [55], we exemplify how to use a zk-SNARK to prove that the attributes committed in the blockchain (and certified by the CA) correspond to a user who satisfies a certain statement. We are interested if the user is born before a year of interest. In the first case, we assumed that each identity attribute is separately hashed and certified in the blockchain. For demonstrating purposes, in *Program 1*, we only used the *year of birth* attribute, but adding more attributes to the scheme can be done similarly. In the given example, *pub_id* is the public identifier of the user (which, for our example is just 128-bits long but can be easily expanded), *(pub_year1, pub_year2)* is the value H_A, certified and stored in the blockchain, which is further compared to the hash *hash_year* computed on the private inputs (the private random value *rand* is added to eliminate immediate brute force attacks on the year of birth, if *pub_id* is small enough). Finally, the validity of the statement is checked. In the second case, we assume that a single certificate is issued for all the user's attributes. For demonstrating purposes, we assume that the user has only two attributes: the year of birth (*year*) and a personal identification number (*id*).

One can add more attributes, such as the ones from Sect. 5.2 [55]. The ZoKrates code that acts as basics for the general proof is illustrated in *Program 2*. The difference between the code in *Program 1* and *Program 2* is that in the latter the sha256 is applied to all the attributes at once, whereas in the first example, each attribute will require the storage of a hash and a certification in the blockchain. We will discuss more about this in the next session.

Table 3 shows the measurements for the exemplified zk-SNARKs programs given before. We used Remix environments for deploying and running blockchain transactions: Javascript Ethereum Virtual Machine (VM) and Injected Web3 (with Metamask Ethereum wallet, a Google Chrome extension [17]). Remix makes a distinction between *execution* and *transaction* costs: execution costs are the costs used on the virtual machine without deployment costs or costs related to function calling, whereas transaction costs include the execution costs as well as the cost of sending contracts and data to the blockchain. When executing transactions using a testnet on Remix, the transactions can be viewed on Etherscan [23], and there are other gas-related measurement fields available, such as the *gas limit* (i.e., the maximum gas amount that can be used in a function execution) and the *gas used* (i.e., the actually used gas amount). Taking this into consideration, for organizing reasons, we choose to represent them together in the last four lines of Table 3, with the mention that the significance of the value depends on the used environment: execution or transaction cost if the environment is Javascript Ethereum VM and gas used or gas limit if the environment is Injected Web3 over Kovan public testnet network.

The setup and the proof generation time do not require Ethereum transactions. They are performed by the ZoKrates plugin and, therefore, have the same values irrespective of the Remix environments. We note that the setup generation time is indeed the highest computational step of the zk-SNARK generation process, but the time for generating a proof is quite acceptable (in the settings mentioned before, using an interface).

We deployed the verifier smart contracts on Kovan testnet, available at [43,44]. The cost of deploying the verifier depends on the environment, and not on the

Table 3. ZoKrates Remix plugin generation and verification measurements

	Program 1		Program 2	
Setup (seconds)	187		237	
Proof (seconds)	70		90	
Environment	Javascript Ethereum VM	Injected Web3 (Kovan network)	Javascript Ethereum VM	Injected Web3 (Kovan network)
Trans. Cost/Gas Limit Deploy Verifier	1299416 gas	1060320 gas	1299148 gas	1060104 gas
Exec. Cost/Gas Used Deploy Verifier	933368 gas	1060320 gas	933168 gas	1060104 gas
Trans. Cost/Gas Limit *verifyTx*	287032 gas	–	286904 gas	–
Exec. Cost/Gas Used *verifyTx*	245280 gas	–	245280 gas	–

proposed example, which is expected as the syntax for the smart contract is the same, and the internal parameters differ slightly. Also, it seems that the gas limit for deploying the smart contract into a public testnet is less than the transaction cost for deploying the verifier on the other environment.

6 Results and Discussion

The verifier smart contract provides a public function called *verifyTx* that can be executed to verify the correctness of a proof. For the Ethereum VM environment, the execution cost for calling this function is the same, irrespective of the program. For the Kovan network, at the moment of writing, we were not able to determine the gas limit and the gas used for performing such transactions, as there seemed to be a bug in the Remix IDE. For the testnet, the transaction fees (calculated as gas price multiplied by the gas used, where the gas price, for our case, equals 9 Gwei) at the exchange from October, 3rd 2020 are 0.00954288 Ether for the first program, respectively 0.009540936 Ether for the second one (2.84 EUR). For smart contract deployments, these are rather acceptable fees that can make more feasible the usage of zk-SNARKs on the blockchain.

Similar experiments were undertaken at the end of June 2020. Then we obtained a significantly larger setup and proof generation time. However, we were able to determine the cost for making *verifyTx* transactions on Kovan testnet: the gas limit for making such transactions was smaller than the transaction cost of making such transactions on the Ethereum VM environment. We noted that the price for smart contract transactions has increased since our previous experiments, therefore, the costs for using such solutions should be taken into consideration. Consequently, we remark the continuous development of the ZoKrates Remix plugin, but also its instability.

On the implementation side, the public user identifier used for the given zk-SNARK programs is of type *field* and can store a maximum of 128 bits (a ZoKrates library constraint). This limitation can cause problems, for example when using an Ethereum public address as a public identifier. However, it can be mitigated by using two parameters of type *field* (an Ethereum address has 256 bits), but one should pay attention to their concatenation.

The considered solution stores the certificates in the blockchain which might be arguable by itself, as certification is by construction publicly verifiable. Despite an overload, certification in blockchain might bring some benefits (e.g., transparency in the sense that anyone can see the certificates and their history). However, the implication of such certification regarding the overall security must be thought of. We assumed certification over a simple hashing on the data, so by using a cryptographically weak hash function the system becomes vulnerable. Brute force attacks caused by a possible small set of values for the attributes (e.g.., possible years of birth) must be mitigated.

Moreover, mechanisms against malleable proof must be considered, preventing an adversary to generate a valid proof, different than, but computed from an eavesdropped valid proof [62]. In the absence of other mechanisms (e.g., verification of correspondence between the committed value on-chain and the executor of the smart contract), the scheme is directly vulnerable to replay attacks: an adversary can fake Alice's identity by reusing a proof that Alice had previously used. Another risk, introduced by construction, resides in the number of users that need to share the proving key generated at setup and used as input together with the witness to generate a valid proof for the deployed smart contract [50].

Depending on the application, the attributes can be individually hashed and certificated or a single hash and one certificate can be used for all attributes. If a single hash is used, the prover is obliged to use all the attributes to generate the proof, regardless of the statement of interest. Adding more parameters to the proof will increase its complexity and therefore the cost of the *verifyTx* will increase. The advantage is in terms of storage space and computation of certifications (constant, regardless of the number of attributes). Choosing what to be stored on the blockchain and the exact form (e.g., the hash, or more general the commitment scheme) remains open to specific application requirements. If sensitive data are stored on-chain, the confidentiality is always at risk, being computationally secured and thus, in time, predisposed to attacks.

Considering the above-mentioned aspects, we highlight that we do not claim the general architecture in Fig. 1 is secure but only use it as an example for our experiments. The motivation or feasibility of applying zk-SNARKs in off-chain settings might be separately investigated. More in-depth implementation details and security analysis of the approach [38] we have considered for our examples, as well as finding better zk-SNARKs solutions or arguing about their utility are outside of the goal of this paper. They remain of interest for future work.

7 Conclusions

Blockchain-based identity management is a domain that might benefit from the usage of zk-SNARKs and zk-STARKs. In this paper, we looked into the practical side of using zk-SNARKs in identity management, by using the ZoKrates library [62] and proposing programs for certain use cases. Although these privacy-preserving technologies seem promising, they are not yet ready to be used in production: the open-source libraries are under development, continuously improved and tested, and still not fully analyzed in terms of security. Improvements in the efficiency of SNARKs and STARKs are also topics for further research.

Acknowledgment. This work was partially supported by a grant of Romanian Ministry of Research and Innovation project no. 17PCCDI/2018.

References

1. Kosba, A.: xJsnark (2020).https://github.com/akosba/xjsnark
2. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006). https://doi.org/10.1007/11693383_22
3. Ben-Sasson, E.: libSTARK (2020). https://github.com/elibensasson/libSTARK
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptology ePrint Archive 2018, 46 (2018)
5. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_16
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 2014), San Diego, CA, pp. 781–796. USENIX Association, August 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson
8. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 326–349 (2012)
9. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC 1988, New York, NY, USA, pp. 103–112. Association for Computing Machinery (1988). https://doi.org/10.1145/62212.62222
10. Bowe, S., Gabizon, A.: Making groth's zk-snark simulation extractable in the random oracle model. IACR Cryptology ePrint Archive 2018, 187 (2018). http://dblp.uni-trier.de/db/journals/iacr/iacr2018.html#BoweG18

11. Bowe, S., Grigg, J., Hopwood, D.: Recursive proof composition without a trusted setup. Technical report, Cryptology ePrint Archive, Report 2019/1021 (2019). https://eprint.iacr.org/2019/1021
12. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. **37**(2), 156–189 (1988)
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 315–334. IEEE (2018)
14. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: modular design and composition of succinct zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2075–2092 (2019)
15. Cerezo Sánchez, D.: Zero-knowledge proof-of-identity: Sybil-resistant, anonymous authentication on permissionless blockchains and incentive compatible, strictly dominant cryptocurrencies. Anonymous Authentication on Permissionless Blockchains and Incentive Compatible, Strictly Dominant Cryptocurrencies, 22 May 2019 (2019)
16. Chiesa, A., Manohar, P., Spooner, N.: Succinct arguments in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11892, pp. 1–29. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_1
17. Consensys: Metamask (2020). https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en
18. Costello, C., et al.: Geppetto: versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy, pp. 253–270 (2015)
19. Dunphy, P., Petitcolas, F.A.: A first look at identity management schemes on the blockchain. IEEE Secur. Privacy **16**(4), 20–29 (2018)
20. Eberhardt, J., Tai, S.: Zokrates-scalable privacy-preserving off-chain computations. In: 2018 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), pp. 1084–1091. IEEE (2018)
21. Edwards, H.: A normal form for elliptic curves. Bull. Am. Math. Soc. **44**(3), 393–422 (2007)
22. Ethereum (2020). https://ethereum.org/en
23. Etherscan: Ethereum Blockchain Explorer (2020). https://etherscan.io
24. Fiat, A., Shamir, A.: How To prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
25. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
26. Gennaro, R., Minelli, M., Nitulescu, A., Orrù, M.: Lattice-based ZK-snarks from square span programs. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 556–573 (2018)
27. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC 1985, New York, NY, USA, pp. 291–304 (1985). Association for Computing Machinery (1985). https://doi.org/10.1145/22145.22178

28. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_19
29. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
30. Groth, J., Maller, M.: Snarky signatures: minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 581–612. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_20
31. GuildOfWeavers: genSTARK (2020). https://github.com/GuildOfWeavers/genSTARK
32. iden3: Circom (2020). https://github.com/iden3/circom
33. iden3: Snarkjs (2020). https://github.com/iden3/snarkjs
34. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, STOC 1992, New York, NY, USA, pp. 723–732. Association for Computing Machinery (1992). https://doi.org/10.1145/129712.129782
35. Kosba, A.: jsnark (2020). https://github.com/akosba/jsnark
36. Lab, S.: libSNARK (2020). https://github.com/scipr-lab/libsnark
37. o1 labs: Snarky (2020). https://github.com/o1-labs/snarky
38. Lee, J., Hwang, J., Choi, J., Oh, H., Kim, J.: Sims: Self sovereign identity management system with preserving privacy in blockchain. IACR Cryptology ePrint Archive 2019, 1241 (2019)
39. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2111–2128 (2019)
40. Meckler, I., Shapiro, E.: Coda: Decentralized cryptocurrency at scale (2018)
41. Micali, S.: Cs proofs. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, SFCS 1994, pp. 436–453. IEEE Computer Society, USA (1994). https://doi.org/10.1109/SFCS.1994.365746
42. Miyaji, A., Nakabayashi, M., Takano, S.: New explicit conditions of elliptic curve traces for FR-reduction. IEICE Trans. Fund. Electron. Commun. Comput. Sci. **84**(5), 1234–1243 (2001)
43. Network, K.T.: Address Smart Contract Program 2 (2020). https://kovan.etherscan.io/address/0x0d0771402acb9d11c73a2df84525b030914a3c47
44. Nework, K.T.: Address Smart Contract Program 1 (2020). https://kovan.etherscan.io/address/0xd7df4c356b182057265a8b36703fb91a9e293b36
45. Nitulescu, A.: Lattice-based zero-knowledge SNARGs for arithmetic circuits. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 217–236. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30530-7_11
46. Github Pages: Zero-Knowledge Proofs (2020). https://zkp.science
47. Panait, A.-E., Olimid, R.F., Stefanescu, A.: Analysis of uPort open, an identity management blockchain-based solution. In: Gritzalis, S., Weippl, E.R., Kotsis, G., Tjoa, A.M., Khalil, I. (eds.) TrustBus 2020. LNCS, vol. 12395, pp. 3–13. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58986-8_1
48. Panait, A.E., Olimid, R.F., Stefanescu, A.: Identity management on blockchain-privacy and security aspects. Proc. Romanian Acad. Ser. A **21**(1), 45–52 (2020)

49. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252 (2013)
50. Pinto, A.: An Introduction to the use of ZK-SNARKs in Blockchains, pp. 233–249, January 2020. https://doi.org/10.1007/978-3-030-37110-4_16
51. Remix: Remix Ethereum-IDE Tool (2019). https://remix.ethereum.org
52. SCIPR Lab: Dizk (2020). https://github.com/scipr-lab/dizk
53. SCIPR Lab: Zexe (2020). https://github.com/scipr-lab/zexe
54. Sharma, B., Halder, R., Singh, J.: Blockchain-based interoperable healthcare using zero-knowledge proofs and proxy re-encryption. In: 2020 International Conference on COMmunication Systems NETworkS (COMSNETS), pp. 1–6 (2020)
55. eIDAS eID Technical Subgroup: eIDAS SAML Attribute Profile (2019). https://ec.europa.eu/cefdigital/wiki/download/attachments/82773108/eIDAS%20SAML%20Attribute%20Profile%20v1.2%20Final.pdf?version=2&modificationDate=1571068651772&api=v2
56. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: DIZK: a distributed zero knowledge proof system. In: 27th USENIX Security Symposium (USENIX Security 2018), pp. 675–692 (2018)
57. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_24
58. Yaga, D., Mell, P., Roby, N., Scarfone, K.: Blockchain technology overview. https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf
59. Zcash: What are zk-SNARKs? (2018). https://z.cash/technology/zksnarks
60. zkcrypto: Bellman (2020). https://github.com/zkcrypto/bellman
61. zkcrypto: Bls12-381 (2020). https://github.com/zkcrypto/bls12_381
62. ZoKrates: Zokrates tutorial (2020). https://zokrates.github.io (All links were last accessed October)