SIMS : Self-Sovereign Identity Management System with Preserving Privacy in Blockchain

Jeonghyuk Lee[a,1], Jungyeon Hwang[c], Jaekyung Choi[b], Hyunok Oh[a,*] and Jihye Kim[b,*]

[a]Hanyang University, Seoul, Korea
[b]Kookmin University, Seoul, Korea
[c] Electronics and Telecommunications Research Institute, Daejeon, Korea

ARTICLE INFO

Keywords:
Self Sovereign Identity Management System
Blockchain
zk-SNARK
Non-interactive argument of knowledge
Commit
Off-chain

ABSTRACT

Blockchain, which is a useful tool for providing data integrity, has emerged as an alternative to centralized servers. Concentrating on the integrity of the blockchain, many applications have been developed. Specifically, a blockchain can be utilized in proving the user's identity using its strong integrity. However, since all data in the blockchain is publicly available, it can cause privacy problems if the user's identity is stored in the blockchain unencrypted. Although the encryption of the private information can diminish privacy problems in the blockchain, it is difficult to transparently utilize encrypted user information in the blockchain. To provide integrity and privacy of user information simultaneously in the blockchain, we propose a SIMS (Self-Sovereign Identity Management System) framework based on a zk-SNARK (zero-knowledge Succinct Non-interactive ARgument of Knowledge). In our proposed SIMS, the user information is employed in a privacy-preserving way due to the zero-knowledge property of the zk-SNARK. We construct a SIMS scheme and prove its security. We describe applications of SIMS and demonstrate its practicality through efficient implementations.

1. Introduction

Blockchain Nakamoto et al. (2008); Wood (2014), which is a useful tool providing data integrity, has emerged as an alternative to centralized servers. Since its construction is based on hash chains where a block references a hash value of the previous block, it is difficult for adversaries to fabricate the previous block value unless malicious users can break a one-way property of the hash function. By utilizing a strong integrity of the blockchain, many blockchain applications have been introduced in cryptocurrencies (Duffield and Diaz, 2015; Miers et al., 2013; Nakamoto et al., 2008; Pilkington, 2016; Wood, 2014), supply chain applications (Pilkington, 2016), and data storage systems (Kshetri, 2017; Zyskind et al., 2015).

Identity proving systems with the blockchain are in the spotlight recently. In these systems, users obtain the information credentials from authorized agents such as the government and upload their credentials into the blockhchain. When an entity such as a service provider has requirements for its customers, users prove the requirements so that they can be verified by the blockchain; the blockchain is used as an transparent infrastructure for identity attestations. In the identity proving system, privacy protection is the most important issue since the information in the block is assumed to be shared (Dhillon et al., 2017). Though Encrypting data and storing the ciphertext in the blockchain may alleviate privacy problems, the issue is how to verify a user's identity from the encrypted data or how to authorize data access. Given the decryption key to the verifier, all possible claims can be verified, but users cannot control their privacy any more. To deal with a privacy-preserving verification problem, we blend the modern zero-knowledge proof system to our proposed scheme.

A zero-knowledge proof system. A zero-knowledge proof system (Campanelli et al.; Costello et al., 2015; Groth, 2010, 2016; Groth et al., 2018; Groth and Maller, 2017; Kosba et al., 2014; Parno et al., 2013; Rackoff and Simon, 1991) enables a user to prove validity of a statement/claim without revealing any additional information.

If the zero-knowledge proof system is adopted in the identity proving system, each user can prove the validity of a statement (i.e. the identity) without revealing detailed personal information. Consider an example that a user wants to prove that her age is greater than 18 years without revealing her actual age. The verifier should only learn that the statement is true but should not learn the user's actual age. Recently, zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARK), which is a non-interactive proof system that provides a constant size proof and fast verification for any general statement, has been actively researched (Campanelli et al.; Costello et al., 2015; Groth, 2016; Groth et al., 2018; Groth and Maller, 2017; Parno et al., 2013). Using the zk-SNARK system, a user has complete control of the way she proves her identity. For instance, when a bank wants to sell a financial product to customers within a certain credit-rate range, a customer who wants to buy the financial product can efficiently provide only her eligibility to purchase the product without revealing her credit rate to the bank through the zk-SNARK approach.

Commit & Prove. The zk-SNARK plays a role of proving the correctness of the identity in the identity proving system. When a commitment scheme is combined with the zk-

*Co-corresponding authors
ahoo791@hanyang.ac.kr (J. Lee); videmot@etri.re.kr (J. Hwang); cjk2889@kookmin.ac.kr (J. Choi); hoh@hanyang.ac.kr (H. Oh); jihyek@kookmin.ac.kr (J. Kim)
ORCID(s):

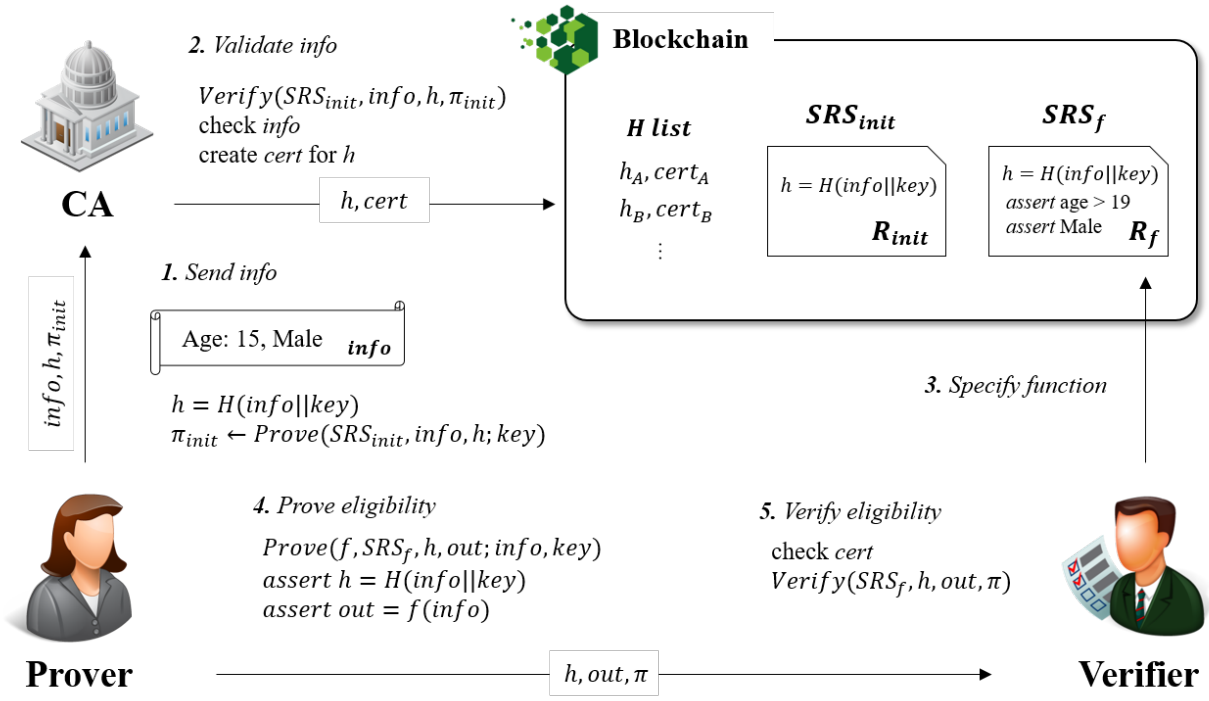

Figure 1: Basic structure of our proposal

SNARK, it can guarantee that the private identity used in zk-SNARK is the same as the identity hidden in the commitment. By proving the commitment is constructed from the private identity and proving that the remaining function output is computed from the same private identity, she can convince the verifier that the function execution result on the identity is correct.

In summary, the blockchain includes a commitment for each user's private information, and the commitment is provided to show the validity of user's information against the given function, which is called a *commit & prove* approach. The commitment plays a role of a public fingerprint in the sense that it is used as an input for the public verification for a specific user.

On-Chain vs. Off-Chain. Blockchain applications can be divided into two types: on-chain blockchain(Kosba et al., 2016; Nakamoto et al., 2008; Sasson et al., 2014; Wood, 2014) and off-chain blockchain(Malavolta et al., 2017; Poon and Dryja, 2016). On-chain utilizes the blockchain as storage for transaction results such as the output of a smart contract or cryptocurrency trade. On the contrary, off-chain exploits the blockchain as a storage for fingerprints and the actual transaction is handled outside of the blockchain Eberhardt and Heiss (2018). Likewise, in the identity proving system, the commitment is stored in the blockchain while the private user information corresponding to the commitment is stored in a private section outside of the blockchain.

Self-sovereign Identity Management System.

This paper proposes a self-sovereign identity management system called SIMS using the commit&prove scheme and the blockchain as a key building block. The concept of the self-sovereign identity is that users have a complete control over how their personal information is kept and used Rouse (2019); Stokkink and Pouwelse (2018); Tobin and Reed (2016). That is, when the users provide an identity to a verifier, the given identity can be verified without any intervention of a third-party who can ensure validity of the identity.

We propose a self-sovereign identity management system using commit&prove and the blockchain. Our approach proceeds as follows.

A user requests an authorized agent to check the user identity and issue a certificate against the identity. The commitment with its certificate are uploaded in the blockchain. The commitment in the blockchain is utilized to prove whether or not the user's confidential data satisfies a given requirement thereby generating a proof. Finally, the proof is verified using the commitment with a valid certificate in the blockchain. Therefore, the commitment in the blockchain is regarded as a digital fingerprint to authenticate the user's information.

Fig. 1 shows an example of our scheme. First of all, when a user wants to upload her identity in the blockchain, the user generates a commitment of her identity `h` using her identity `info` and random secret value `key`. To prove a relation between the user identity `info` and the commitment `h` without revealing the random secret value `key`, the user sends her commitment `h` and identity `info` to the certificated authority with a proof π_{init}. When the certificated authority receives them, it checks the validity of the identity `info` and generates a certificate on the commitment. Then the commitment and its certificate are uploaded to the blockchain by the cer-

tificated authority. Apart from the commitment generation process, a verifier such as an organization, bank, corporation, etc. provides information processing function f and its shared reference string $\mathsf{SRS_f}$. The verifier uploads them to the blockchain. Later, the user proves that her private identity satisfies a given function by generating a zk-SNARK. Then the proof is verified with the commitment and the certificate in the blockchain against the given function.

Our Contributions. We describe our contributions as follows:

- **Defining a notion of self-sovereign identity management system :** We define the concept of self-sovereign identity management system in the blockchain environment. We also define a security notion of self-sovereign identity management system which can be applied to any blockchain environment. Our security notion relies on a simulation-based approach that ensures a privacy property and a soundness property concurrently.
- **Constructing an efficient privacy preserving identity management scheme** :We provide a self-sovereign identity management construction SIMS which can be implemented by any zk-SNARK and general blockchain such as Dhillon et al. (2017); Nakamoto et al. (2008); Wood (2014).
- **Providing a security proof of the security notion :** We prove that the proposed SIMS is secure against the security model. We simulate an adversary in an ideal environment where a trusted party who plays the role of a cryptographic protocol in a real system exists. Then we show that an adversary in the real protocol has an ability to forge a function output at most equal to that of the adversary in the ideal environment.
- **Providing practical experimental results :** We demonstrate realistic and practical self-sovereign identity management applications. The proposed applications were implemented based on (Kosba et al., 2018) as the front-end, and libsnark (Ben-Sasson et al.) as the back-end. The experiment result validates the practicality of the proposed SIMS. Specifically, the proof time of the simple check function takes only 3.8ms and all applications have a constant proof size regardless of their proof time.

We describe the background in section 2. In section 3, we describe the specific construction. Section 4 presents the security proof of our construction. We discuss the applications of our proposal in section 5.1. Section 5 provides experiment results of our applications. In section 6, we describe related work of our proposal and we draw a conclusion in section 7.

2. Background

2.1. Notation

Let $\mathcal{R}$ be a relation generator that given a security parameter λ in unary returns a polynomial time decidable relation $\mathsf{R} \leftarrow \mathcal{R}(1^\lambda)$. For $(\phi, \mathsf{w}) \in \mathsf{R}$ we call ϕ the instance and w the witness. We define $\mathcal{R}_\lambda$ to be the set of possible relations $\mathcal{R}(1^\lambda)$ might output.
We write $y \leftarrow S$ for sampling y uniformly at random from the set S. We write $y \leftarrow A(x)$ for a probabilistic algorithm on input x returning output y. We use the abbreviation PPT for probabilistic polynomial time algorithms. For an algorithm A we define $trans_A$ to be a list containing all of A's inputs and outputs, including random coins. We define a specific NIZK algorithm in section 2.2 which comes from (Groth and Maller, 2017).

2.2. Non-interactive Zero-Knowledge Arguments of Knowledge

Definition 2.1. *Let $\mathcal{R}$ be a relation generator. A NIZK argument for $\mathcal{R}$ is a quadruple of algorithms* (Setup, Prove, Verify, SimProve), *which is complete, zero-knowledge, and knowledge sound and works as follows:*

- $(\mathsf{srs}, \tau) \leftarrow \mathsf{Setup}(\mathsf{R})$ *: the setup algorithm is a PPT algorithm which takes a relation* $\mathsf{R} \in \mathcal{R}_\lambda$ *as an input and returns a shared reference string* srs *and a simulation trapdoor* τ.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{srs}, \phi; \mathsf{w})$ *: the prover algorithm is a PPT algorithm which takes a shared reference string* srs *for a relation* R *and* $(\phi, \mathsf{w}) \in \mathsf{R}$ *as inputs and returns a proof* π.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{srs}, \phi, \pi)$ *: the verifier algorithm is a deterministic polynomial time algorithm which takes a shared reference string* srs *and an instance* ϕ *and a proof* π *as inputs and returns 0 (reject) or 1 (accept).*
- $\pi \leftarrow \mathsf{SimProve}(\mathsf{srs}, \tau, \phi)$ *: the simulator is a PPT algorithm which takes a shared reference string* srs*, a simulation trapdoor* τ *and an instance* ϕ *as inputs and returns a proof* π.

Perfect Completeness : Perfect completeness says that given a true statement, a prover with a witness can convince the verifier.

Definition 2.2. (Setup, Prove, Verify, SimProve) *is a perfectly complete argument system for* R *if for all* $\lambda \in N$ *for all* $\mathsf{R} \in \mathcal{R}_\lambda$ *and for all* $(\phi, \mathsf{w}) \in \mathsf{R}$*:*

$$\Pr[(\mathsf{srs}, \tau) \leftarrow \mathsf{Setup}(\mathsf{R}); \pi \leftarrow \mathsf{Prove}(\mathsf{srs}, \phi, \mathsf{w}) : \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1] = 1$$

Note that the simulation trapdoor τ is kept secret and is not known to either prover or verifier in normal use of the NIZK argument, but it enables the simulation of proofs when we define zero-knowledge below.

Perfect Zero-Knowledge : An argument system has a perfect zero-knowledge if it does not leak any information besides the truth of the instance. This is modelled a simulator that does not know the witness but has some trapdoor information that enables it to simulate proofs.

Definition 2.3. *For* $Arg = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimProve})$ *an argument system, define* $Adv^{zk}_{Arg,A}(\lambda) = 2\Pr[\mathcal{G}^{zk}_{Arg,A}(\lambda)] - 1$ *where the game* $\mathcal{G}^{zk}_{Arg,A}(\lambda)$ *is defined as follows.*

$$\frac{\mathsf{MAIN}\ \mathcal{G}^{zk}_{Arg,A}(\lambda)}{\begin{array}{r}\mathsf{R} \leftarrow \mathcal{R}(1^\lambda)\\ (\mathsf{srs}, \tau) \leftarrow \mathsf{Setup}(\mathsf{R})\\ b \leftarrow \{0,1\}\\ b' \leftarrow A^{P^b_{\mathsf{srs},\tau}}(\mathsf{srs})\\ if\ b = b'\ return\ 1\\ return\ 0\ otherwise\end{array}}$$

$$\frac{P^0_{\mathsf{srs},\tau}(\phi_i, w_i)}{\begin{array}{r}assert(\phi_i, \mathsf{w}_i) \in \mathsf{R}\\ \pi_i \leftarrow \mathsf{Prove}(\mathsf{srs}, \phi, \mathsf{w})\\ return\ \pi_i\end{array}} \qquad \frac{P^1_{\mathsf{srs},\tau}(\phi_i, w_i)}{\begin{array}{r}assert(\phi_i, \mathsf{w}_i) \in \mathsf{R}\\ \pi_i \leftarrow \mathsf{SimProve}(\mathsf{srs}, \tau, \phi)\\ return\ \pi_i\end{array}}$$

The argument system Arg is perfectly zero knowledge if all PPT adversaries $Adv^{zk}_{Arg,A}(\lambda) = 0$

Computational Knowledge Soundness : An argument system is computationally knowledge sound if whenever somebody produces a valid argument it is possible to extract a valid witness from their internal data.

Definition 2.4. *For* $Arg = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimProve})$ *an argument system, define* $Adv^{sound}_{Arg,A,\chi_A}(\lambda) = \Pr[\mathcal{G}^{sound}_{Arg,A,\chi_A}(\lambda)]$ *where the game* $\mathcal{G}^{sound}_{Arg,A,\chi_A}(\lambda)$ *is defined as follows.*

$$\frac{\mathsf{MAIN}\ \mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_A}(\lambda)}{\begin{array}{r}\mathsf{R} \leftarrow \mathcal{R}(1^\lambda)\\ (\mathsf{srs}, \tau) \leftarrow \mathsf{Setup}(\mathsf{R})\\ (\phi, \pi) \leftarrow \mathcal{A}(\mathsf{srs})\\ \mathsf{w} \leftarrow \chi_A(trans_A)\\ assert\ (\phi, \mathsf{w}) \notin \mathsf{R}\\ return\ \mathsf{Verify}(\mathsf{srs}, \phi, \pi)\end{array}}$$

An argument system Arg is computationally knowledge sound if for any PPT adversary A, there exists a PPT extractor χ_A such that $Adv^{sound}_{Arg,A,\chi_A}(\lambda) \approx 0$.

2.3. Composable zk-SNARK

Our construction has a commitment for a private identity. To commit the private identity of the prover, hash functions (e.g. SHA-256, MD5) are usually utilized. However, if the commitment function is included in the given function by the verifier, then the function becomes relatively large. That is, when a prover generates a proof of the given function, the prover not only computes a proof for processing the function body but also checks whether the commitment value holds h=Hash(x) where x is the private identity of the prover. If a hash function is included, the proving time and srs size increase. To overcome the problem, we use a composable zk-SNARK (Fiore et al., 2016). The composable zk-SNARK supports a combined form of the proof where correctness of function execution is handled by the NIZK while data authentication which is related to a commitment computation is handled by a specific proof construction which is executed outside of the proof circuit. Since the proof cost required for inner encoding which is related to the data authenticity can be reduced, the proof computation becomes more efficient. We describe a composable zk-SNARK based on (Fiore et al., 2016).

Before describing the composable zk-SNARK, we extend a notion of the NIZK algorithm to verify the two different proofs separately. One is related to commitment called an offline verification, the other is related to function execution called an online verification. We adopt the definition of **online/offline** verification from (Fiore et al., 2016).

The verification algorithm of many NIZK constructions can be split into offline and online computations. For many NIZK, there exists algorithms $(\mathsf{Online}, \mathsf{Offline})$ such that:

$$\mathsf{Verify}_{\mathsf{NIZK}}(\mathsf{srs}, \phi, \pi) = \mathsf{Online}(\mathsf{srs}, \mathsf{Offline}(\mathsf{srs}, \mathsf{x}), \mathsf{v}, \pi)$$

The offline phase can be seen as the computation of one or more commitment $c_x (c_x = \mathsf{Offline}(\mathsf{srs}, x))$, some of which may be computed by the prover, and possibly never opened by the verifier. In this work, we consider schemes where the Offline computations consist purely of multi-exponentiations in $\mathbb{G}_1$ over the instance ϕ , followed by *online* computations that accept or reject the proof. We assume that $X = \mathbb{Z}_p$ as a domain of a x and $\mathsf{Offline}(\mathsf{srs}, x) = \prod F_i^{x_i}$ from X to $\mathbb{G}_1$, where the group element $(F_1, F_2, ..., F_n) \in {\mathbb{G}_1}^n$ are part of the keys.

An intuition of the composable zk-SNARK is that parts of the proof which are related to data authentication are generated before the proof generation phase, and the parts of the proof that are related to the function execution are generated later in the proof circuit. We call the proof generation phase executed outside the proof circuit as XP algorithm. XP algorithm is defined as follows.

$\mathsf{Setup}_{\mathsf{XP}}(1^\lambda)$ samples $H_i \xleftarrow{\$} \mathbb{G}_1$ for $i \in [1, n]$ and returns $\mathsf{pp} = (\mathcal{G}_\lambda, H)$ where $H = (H_1, ..., H_n)$.

$\mathsf{Hash}_{\mathsf{XP}}(pp, (x_1, ..., x_n))$ returns $\sigma_x \leftarrow \prod_{i\in[1,n]} H_i^{x_i}$

$\mathsf{KeyGen}_{\mathsf{XP}}(\mathsf{pp}, \mathsf{F})$ samples $u, v, w \xleftarrow{\$} \mathbb{Z}_p$ and computes $U \leftarrow g_2^u, V \leftarrow g_2^v, W \leftarrow g_2^w$; samples $R_i \xleftarrow{\$} \mathbb{G}_1$ and computes $T_i \leftarrow H_i^u R_i^v F_i^w$ for $i \in [1, n]$; and returns $\mathsf{EK}_\mathsf{F} = (F, T, R)$ and $\mathsf{VK}_\mathsf{F} = (U, V, W)$ where $R = (R_1, ..., R_n)$ and $T = (T_1, ..., T_n)$.

$\mathsf{Prove}_{\mathsf{XP}}(\mathsf{EK}_\mathsf{F}, (x_1, ..., x_n), c_x)$ computes $T_x \leftarrow \prod_{i\in[1,n]} T_i^{x_i}$

and $R_x \leftarrow \prod_{i\in[1,n]} R_i^{x_i}$; and returns $\Phi_x = (T_x, R_x)$.
(implicitly we require that $c_x = \prod_{i\in[1,n]} F_i^{x_i}$, though the c_x part of the instance is not used in the computation of the proof.)

$\mathsf{Verify}_{\mathsf{XP}}(\mathsf{VK}_{\mathsf{F}}, \sigma_x, c_x, \Phi_x)$ parses $\Phi_x = (T_x, R_x)$ and returns

$$e(T_x, g_2) \stackrel{?}{=} e(\sigma_x, U)e(R_x, V)e(c_x, W).$$

Through the XP algorithm, a proof computation required for the data authenticity such as a commitment verification process can be extracted from the proof circuit. Parts of the proof computation which are not related to the function execution can be computed in advance since the commitment algorithm Hash in XP has a specific form that is exponentiation of the group element and can be materials of the proof. Specifically, the evaluation key EK_{F} has a group element H which is a base of the commitment and a proof can be computed by only some exponential computation.

We describe a composable zk-SNARK which supports a general function using a combination of the XP and the NIZK in Fiore et al. (2016). In this approach, a proof for the commitment is constructed using the XP and a proof for a function execution is generated using the NIZK algorithm. And we assume that XP algorithm can supports any NIZK algorithm if XP and NIZK algorithm have the same public parameters such as the same bilinear group setting.
Composable zk-SNARK is defined as follows.

$\mathsf{Setup}(1^\lambda)$ runs $\mathsf{Setup}_{\mathsf{XP}}(1^\lambda)$ and returns its public parameters pp.

$\mathsf{Hash}(\mathsf{pp}, \mathsf{x})$ returns $\sigma_x := \mathsf{Hash}_{\mathsf{XP}}(\mathsf{pp}, \mathsf{x})$

$\mathsf{KeyGen}(\mathsf{pp}, \mathsf{R})$ takes a relation R and runs

$(\mathsf{EK}_{\mathsf{NIZK}}, \mathsf{VK}_{\mathsf{NIZK}}) \leftarrow \mathsf{Setup}_{\mathsf{NIZK}}(\mathsf{R})$
Let $\mathsf{F} =: (F_1, F_2, ..., F_n)$ be the offline element in $\mathsf{VK}_{\mathsf{NIZK}}$
$(\mathsf{EK}_{\mathsf{XP}}, \mathsf{VK}_{\mathsf{XP}}) \leftarrow \mathsf{Keygen}_{\mathsf{XP}}(\mathsf{pp}, \mathsf{F})$
return $\mathsf{srs} := (\mathsf{EK}_{\mathsf{NIZK}}, \mathsf{VK}_{\mathsf{NIZK}}, \mathsf{EK}_{\mathsf{XP}}, \mathsf{VK}_{\mathsf{XP}})$

$\mathsf{Prove}(\mathsf{srs}, \phi; \mathsf{w})$ parses srs then runs

$c_x \leftarrow \mathsf{Offline}_{\mathsf{NIZK}}(\mathsf{srs}, x)$ where $\phi = (x, \upsilon)$
$\pi_{\mathsf{NIZK}} = \mathsf{Prove}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}}, \phi; \mathsf{w})$
$\pi_{\mathsf{XP}} = \mathsf{Prove}_{\mathsf{XP}}(\mathsf{srs}_{\mathsf{XP}}, x, c_x)$
return $\pi := (c_x, \pi_{\mathsf{NIZK}}, \pi_{\mathsf{XP}})$

$\mathsf{Verify}(\mathsf{srs}, \sigma_x, \upsilon, \pi)$ parses srs as $(\mathsf{EK}_{\mathsf{NIZK}}, \mathsf{VK}_{\mathsf{NIZK}}, \mathsf{EK}_{\mathsf{XP}}, \mathsf{VK}_{\mathsf{XP}})$, π as $(c_x, \pi_{\mathsf{NIZK}}, \pi_{\mathsf{XP}})$ and returns

$$\mathsf{Online}_{\mathsf{NIZK}}(\mathsf{srs}, c_x, \upsilon, \pi_{\mathsf{NIZK}}) \wedge \mathsf{Verify}_{\mathsf{XP}}(\mathsf{srs}, \sigma_x, c_x, \pi_{\mathsf{XP}})$$

As described above, the composable zk-SNARK has a proof which is a combined form. In the Prove, the prover creates the final proof by combining c_x that is related to the function I/O, π_{NIZK} which is the proof of the function's witness, and the proof of the commit calculation π_{XP}. In the Verify, the verifier verifies π_{NIZK} and π_{XP} respectively. Since π_{XP} and c_x can be generated by a pre-computation, the composable zk-SNARK can reduce the proof generation cost.

2.4. Definition

The proposed SIMS supposes three entities. these entities are as follows.

- Prover : It is a user holding a private identity. And it provides Verifier with a function output that is computed from the private identity.
- CA (Certificated Authority) : It is the authorized agent that provides the authenticity of user's private identity. It registers the commitment of the private identity with its certificate in the blockchain. CA can be a government department which manages the private identity that requires validity or an organization that issues certificates for an individual.
- Verifier : It verifies whether the function output which is provided from Prover satisfies the requested function.

Our scheme consists of five algorithms as follows.

- $\mathsf{Init}(\mathsf{info}, \mathsf{key}, \mathsf{srs}_{\mathsf{init}})$: It is an initialization algorithm for registering a private identity. Prover takes its private identity info, random secret value key, shared reference string $\mathsf{srs}_{\mathsf{init}}$ as inputs. It computes a commitment of the private user identity h then generates its proof π_{init}. It returns h, info, π_{init}.
- $\mathsf{Register}(\mathsf{L}, \mathsf{h}, \mathsf{info}, \pi_{\mathsf{init}}, \mathsf{srs}_{\mathsf{init}})$: CA takes a blockchain L, a commitment of user's private identity h, user's private identity info, a proof π_{init} , shared reference string $\mathsf{srs}_{\mathsf{init}}$ as inputs. It computes a certificate on commitment h, then uploads the commitment h, certificate cert in the blockchain L. Only the commitment of the private identity is available to the public while the private identity info is kept in user's private section.
- $\mathsf{Setup}(f)$: Verifier takes a function f as an input. It computes srs for f and then returns srs.
- $\mathsf{Prove}(\mathsf{info}, \mathsf{key}, \mathsf{h}, f, \mathsf{srs})$: Prover takes a private identity info, secret random value key, commitment h, function f, shared reference string srs as inputs. It returns function output out and its proof π.
- $\mathsf{Verify}(\pi, \mathsf{h}, \mathsf{out}, f, \mathsf{srs})$: Verifier takes a proof π, commitment h, function output out, shared reference string srs as inputs. It verifies π then returns 0 (reject) or 1(accept).

2.5. Security model

We define a security model of the privacy preserving self-sovereign identity management scheme in this section.

The security model follows the simulation based definition (Garman et al., 2016). We assume an ideal world functionality where any adversary cannot fabricate function output and distinguish a commitment, and then we show that the probability of an adversary compromising a real world execution is at most equal to the probability of an adversary compromising an ideal world execution at best. To correlate the adversary in the real world execution with the adversary in the ideal world execution, we show that output of the adversary in the ideal world execution is computationally indistinguishable from the output of the adversary in the real world execution. And we prove that the ideal world execution and the real world execution computationally have same distributions.

We define the ideal world execution which uses the ideal functionality interacting with a trusted party TP that plays a role of our cryptographic constructions in the real execution.

Ideal world execution Ideal: The ideal world attacker, S, selects some subset of the $P_1, \ldots, P_n$ parties to corrupt and informs the TP. While the attacker controlled parties behave arbitrarily, the honest ones follow a set of strategies Σ. All parties then interact via messages passed to and from each subject implementing the ideal functionality outlined in Algorithm 3.

Real world execution Real : The real world attacker $\mathcal{A}$ controls a subset of the parties $P_1, \ldots, P_n$ interacting with the real functionality. The honest parties execute the commands output by the strategy Σ using the real functionality while the attacker controlled parties can behave arbitrarily. We assume that all parties can interact with a trusted append only blockchain.

Definition 2.5. *We say that real functionality SIMS securely emulates the ideal functionality provided by* TP *if for all probabilistic polynomial-time real world adversaries $\mathcal{A}$ and all honest party strategies Σ, there exists a simulator $\mathcal{S}$ such that for any ppt distinguisher d:*

$$\mathsf{P}[\mathsf{d}(\mathsf{Ideal}_{\mathsf{TP},\mathcal{S},\Sigma}(\lambda)=1]-\mathsf{P}[\mathsf{d}(\mathsf{Real}_{\mathsf{SIMS},\mathcal{A},\Sigma}(\lambda))=1]\leq \mathsf{negl}(\lambda)$$

Theorem 1. *SIMS satisfies with Definition 2.5 given the existence of Non-interactive Zero-Knowledge Argument of Knowledge, statistically hiding and computationally binding commitments.*

A detailed proof of Theorem 1 is described in section 4.

3. Construction

We describe a specific construction of our SIMS in this section.

Algorithm 1 explains the proposed SIMS in details. Assume that $\mathsf{NIZK} = (\mathsf{Setup}_{\mathsf{NIZK}}, \mathsf{Prove}_{\mathsf{NIZK}}, \mathsf{Verify}_{\mathsf{NIZK}})$ is a non-interactive zero-knowledge scheme that supports a sequence of relations R.

In Init, Prover takes a private identity info, random value key for randomizing commitment of the private identity and shared reference string $\mathsf{srs}_{\mathsf{init}}$ as inputs. Prover computes a commitment h for the private identity info and random secret value key. Then, Prover runs NIZK prove algorithm $\mathsf{Prove}_{\mathsf{NIZK}}$ to prove correctness of commitment h and acquires a proof π_{init}. Prover sends $(\mathsf{info}, \mathsf{h}, \pi_{\mathsf{init}})$ to CA. We assume that a secret random value that is used to generate a commitment h is not revealed to anyone including CA. Only Prover has a secret value key, which restricts information management privilege to Prover. However, for the validity of the private information info, it needs to be certified from the certificated authority. Thus, Prover shares $(\mathsf{info}, \mathsf{h})$ with CA to certify the private identity info without revealing secret value key. To prove a relation between the private identity info and the commitment h, Prover provides the NIZK proof π_{init} to CA.

In Register, CA takes a blockchain L, prover's private identity info, commitment h, proof π_{init}, shared reference string $\mathsf{srs}_{\mathsf{init}}$ as inputs. It first verifies a proof π_{init} that proves a statement where the commitment h is computed from the private identity info. If it is verified, CA computes a certificate of the commitment h cert. Finally CA records $(\mathsf{h}, \mathsf{cert})$ to the

Algorithm 1 SIMS

PublicSetup($\cdot$)
- $\mathsf{R}_{\mathsf{init}} = \{(\phi, \mathsf{w}) | \phi = (\mathsf{info}, \mathsf{h}), \mathsf{h} = \mathsf{Hash}(\mathsf{info}||\mathsf{key})\}$
- $\mathsf{srs}_{\mathsf{init}} \leftarrow \mathsf{Setup}_{\mathsf{NIZK}}(\mathsf{R}_{\mathsf{init}})$
- return $\mathsf{srs}_{\mathsf{init}}$

Init($\mathsf{info}, \mathsf{key}, \mathsf{srs}_{\mathsf{init}}$)
- $\mathsf{h} \leftarrow \mathsf{Hash}(\mathsf{info}||\mathsf{key})$
- $\pi_{\mathsf{init}} \leftarrow \mathsf{Prove}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{init}}, \phi_{\mathsf{init}}; \mathsf{w}_{\mathsf{init}})$
  - where $\phi_{\mathsf{init}} = (\mathsf{info}, \mathsf{h})$
- sends $(\mathsf{info}, \mathsf{h}, \pi_{\mathsf{init}})$ to CA

Register($\mathsf{L}, \mathsf{info}, \mathsf{h}, \pi_{\mathsf{init}}, \mathsf{srs}_{\mathsf{init}}$)
- $\mathsf{b} \leftarrow \mathsf{Verify}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{init}}, \phi_{\mathsf{init}}, \pi_{\mathsf{init}})$
  - where $\phi_{\mathsf{init}} = (\mathsf{info}, \mathsf{h})$
- **if** $\mathsf{b} = 1$ **then**
  - computes certificate of h cert
- **else**
  - abort
- **end if**
- records $(\mathsf{h}, \mathsf{cert})$ to the blockchain L

Setup(f)
- $\mathsf{R} = \{(\phi, \mathsf{w}) | \phi = (\mathsf{h}, \mathsf{out}), \mathsf{out} = f(\mathsf{info}), \mathsf{h} = \mathsf{Hash}(\mathsf{info}||\mathsf{key})\}$
- $\mathsf{srs} \leftarrow \mathsf{Setup}_{\mathsf{NIZK}}(\mathsf{R})$
- return srs

Prove($\mathsf{info}, \mathsf{key}, \mathsf{h}, f, \mathsf{srs}$)
- $\mathsf{out} \leftarrow f(\mathsf{info})$
- $\pi \leftarrow \mathsf{Prove}_{\mathsf{NIZK}}(\mathsf{srs}, \phi; \mathsf{w})$
  - where $\phi = (\mathsf{h}, \mathsf{out})$ and $\mathsf{w} = (\mathsf{info}, \mathsf{key})$
- return (out, π)

Verify($\pi, \mathsf{h}, \mathsf{out}, \mathsf{srs}$)
- $\mathsf{b} \leftarrow \mathsf{Verify}_{\mathsf{NIZK}}(\mathsf{srs}, \phi, \pi)$ where $\phi = (\mathsf{h}, \mathsf{out})$
- return b

blockchain L. We suppose that the certificate can be generated by a signature scheme which satisfies a security property of unforgeability against chosen-message attacks and cert can be publicly verifiable. Only the commitment of the private identity is revealed to the public and the private identity info is not available publicly.

In Setup, Verifier takes a function f as an input. It constructs a new circuit containing the given fuction f and a commit relation Hash where a relation $\mathsf{R} = \{(\phi, \mathsf{w}) | \phi = (\mathsf{h}, \mathsf{out}), \mathsf{out} = f(\mathsf{info}), \mathsf{h} = \mathsf{Hash}(\mathsf{info}||\mathsf{key})\}$. Then it calls NIZK setup algorithm $\mathsf{Setup}_{\mathsf{NIZK}}$ for relation R, receives srs, and returns srs.

In Prove, Prover takes the commitment h, private identity info, random value key, function f, and shared reference string srs as inputs. It executes a function f with providing info to obtain result out. And then, it calls a NIZK prove algorithm $\mathsf{Prove}_{\mathsf{NIZK}}$ with input (h, out) and witness (info, key) and acquires a proof π.

Verifier takes a proof π, commitment h, function output out, shared reference string srs, as inputs in Verify. It first checks a validity of the commitment, and calls $\mathsf{Verify}_{\mathsf{NIZK}}$ which is a proof verification algorithm to guarantee that the function input and output are correct, and returns the result.

We present the proposed SIMS in detail with the commitment described in section 2.3. Using the commit-and-prove scheme in (Fiore et al., 2016), we can reduce the overhead of the commitment generation in the proof generation. Specifically, we construct a commitment value h using XP algorithm in section 2.3. Since we generate a partial proof for commitment relation outside of the proving process, we can prove the commitment relation by applying some exponential operation. We construct a proof for information processing function f and a proof for the commitment, respectively. In Verify, the proof verification process is divided into two processes, which are the verification for the function correctness and commitment correctness as described in section 2.3.

Algorithm 2 describes a specific construction of optimized SIMS. In Init, Prover takes a public parameter for the commitment pp, private identity info, secret value key, shared reference string $\mathsf{srs}_{\mathsf{init}}$ as inputs. To use XP algorithm for the commitment generation, it uses the public parameter pp from $\mathsf{Setup}_{\mathsf{XP}}$. After that, it runs $\mathsf{Hash}_{\mathsf{XP}}$ taking pp, info||key as inputs and acquires commitment value h. Instead of applying commitment algorithms such as SHA-256, MD5, etc., we adopt $\mathsf{Hash}_{\mathsf{XP}}$ algorithm whose commitment form is similar to pederson commitment (Di Crescenzo et al., 2001). To prove correctness of the commitment, it runs $\mathsf{Offline}_{\mathsf{NIZK}}$ and gets the partial proof of I/O parts $\mathsf{c}_{\mathsf{x}_{\mathsf{init}}}$. Then it runs $\mathsf{Prove}_{\mathsf{NIZK}}$ algorithm and takes a proof for the witness $\pi_{\mathsf{NIZK}_{\mathsf{init}}}$. It operates $\mathsf{Prove}_{\mathsf{XP}}$ and gets a proof for the commitment computation $\pi_{\mathsf{XP}_{\mathsf{init}}}$. Finally, it sends the private identity info, the commitment h and all the proofs $(\pi_{\mathsf{init}} = \mathsf{c}_{\mathsf{x}_{\mathsf{init}}}, \pi_{\mathsf{XP}_{\mathsf{init}}}, \pi_{\mathsf{NIZK}_{\mathsf{init}}})$ to CA. Though its construction is similar to proof generation parts of the commitment in Prove algorithm, it has a difference in the sense that the proof in the Init algorithm reveals the private identity info while a proof in the Prove algorithm does not reveal the private identity.

Algorithm 2 SIMS optimization

PublicSetup($\cdot$)
- $\mathsf{pp} \leftarrow \mathsf{Setup}_{\mathsf{XP}}(1^{\lambda})$
- $\mathsf{R}_{\mathsf{init}} = \{(\phi_{\mathsf{init}}, \mathsf{w}_{\mathsf{init}}) | \phi_{\mathsf{init}} = (\mathsf{info}, \mathsf{h})\}$
- $\mathsf{srs}_{\mathsf{NIZK}_{\mathsf{init}}} \leftarrow \mathsf{Setup}(\mathsf{R}_{\mathsf{init}})$
- $\mathsf{srs}_{\mathsf{XP}_{\mathsf{init}}} \leftarrow \mathsf{Keygen}_{\mathsf{XP}}(\mathsf{pp}, \mathsf{srs}_{\mathsf{NIZK}_{\mathsf{init}}})$
- $\mathsf{srs}_{\mathsf{init}} = (\mathsf{srs}_{\mathsf{NIZK}_{\mathsf{init}}}, \mathsf{srs}_{\mathsf{XP}_{\mathsf{init}}})$
- return $(\mathsf{pp}, \mathsf{srs}_{\mathsf{init}})$

Init(pp, info, key, $\mathsf{srs}_{\mathsf{init}}$)
- $\mathsf{h} \leftarrow \mathsf{Hash}_{\mathsf{XP}}(\mathsf{pp}, \mathsf{info}||\mathsf{key})$
- $\mathsf{c}_{\mathsf{x}_{\mathsf{init}}} \leftarrow \mathsf{Offline}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}_{\mathsf{init}}}, \mathsf{info}||\mathsf{key})$
- $\pi_{\mathsf{NIZK}_{\mathsf{init}}} \leftarrow \mathsf{Prove}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}_{\mathsf{init}}}, \phi_{\mathsf{init}})$
  - where $\phi_{\mathsf{init}} = (\mathsf{info}, \mathsf{h})$
- $\pi_{\mathsf{XP}_{\mathsf{init}}} \leftarrow \mathsf{Prove}_{\mathsf{XP}}(\mathsf{srs}_{\mathsf{XP}_{\mathsf{init}}}, \mathsf{info}||\mathsf{key})$
- $\pi_{\mathsf{init}} = (\mathsf{c}_{\mathsf{x}_{\mathsf{init}}}, \pi_{\mathsf{XP}_{\mathsf{init}}}, \pi_{\mathsf{NIZK}_{\mathsf{init}}})$
- sends $(\mathsf{info}, \mathsf{h}, \pi_{\mathsf{init}})$ to CA

Register(L, info, h, π_{init}, $\mathsf{srs}_{\mathsf{init}}$)
- $\mathsf{b} \leftarrow \mathsf{Online}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}_{\mathsf{init}}}, \mathsf{c}_{\mathsf{x}_{\mathsf{init}}}, \mathsf{h}, \pi_{\mathsf{NIZK}_{\mathsf{init}}}) \wedge \mathsf{Verify}_{\mathsf{XP}}(\mathsf{srs}_{\mathsf{XP}_{\mathsf{init}}}, \mathsf{h}, \mathsf{c}_{\mathsf{x}_{\mathsf{init}}}, \pi_{\mathsf{XP}_{\mathsf{init}}})$
- **if** b = 1 **then**
  - computes a certificate of h cert
- **else**
  - abort
- **end if**
- records (h, cert) to the blockchain L

Setup(f)
- $\mathsf{R}_{\mathsf{NIZK}} = \{(\phi, \mathsf{w}) | \phi = (\mathsf{h}, \mathsf{out}), \mathsf{out} = f(\mathsf{info})\}$
- $\mathsf{srs}_{\mathsf{NIZK}} \leftarrow \mathsf{Setup}_{\mathsf{NIZK}}(\mathsf{R}_{\mathsf{NIZK}})$
- $\mathsf{srs}_{\mathsf{XP}} \leftarrow \mathsf{KeyGen}_{\mathsf{XP}}(\mathsf{pp}, \mathsf{srs}_{\mathsf{NIZK}})$
- $\mathsf{srs} = (\mathsf{srs}_{\mathsf{NIZK}}, \mathsf{srs}_{\mathsf{XP}})$
- return srs

Prove(info, key, h, f, srs)
- $\mathsf{out} \leftarrow f(\mathsf{info})$
- $c_x \leftarrow \mathsf{Offline}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}}, \mathsf{info}||\mathsf{key})$
- $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{Prove}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}}, \phi)$
  - where $\phi = (\mathsf{info}, \mathsf{key}, \mathsf{out})$
- $\pi_{\mathsf{XP}} \leftarrow \mathsf{Prove}_{\mathsf{XP}}(\mathsf{srs}_{\mathsf{XP}}, \mathsf{info}||\mathsf{key})$
- $\pi = (c_x, \pi_{\mathsf{XP}}, \pi_{\mathsf{NIZK}})$
- return (out, π)

Verify(π, h, out, srs)
- $\mathsf{b} \leftarrow \mathsf{Online}_{\mathsf{NIZK}}(\mathsf{srs}_{\mathsf{NIZK}}, c_x, \mathsf{out}, \pi_{\mathsf{NIZK}}) \wedge \mathsf{Verify}_{\mathsf{XP}}(\mathsf{srs}_{\mathsf{XP}}, \mathsf{h}, c_x, \pi_{\mathsf{XP}})$
- return b

In Register, CA takes a blockchain L, private identity info, its commitment h, proof π_{init}, shared reference string $\mathsf{srs}_{\mathsf{init}}$ as inputs. It first verifies the proof π_{init} taking input $\mathsf{srs}_{\mathsf{init}}$. If the proof is verified, it generates a certificate of h and records it to the blockchain L with its certificate cert.

Setup algorithm generates the shared reference string for information processing and the commitment respectively. To

prove function and commitment correctness, we generate appropriate strings of $\mathsf{srs}_{\mathsf{NIZK}}$ and $\mathsf{srs}_{\mathsf{XP}}$ by performing $\mathsf{Setup}_{\mathsf{NIZK}}$ and $\mathsf{KeyGen}_{\mathsf{XP}}$.

In Prove, Prover takes inputs same as proving algorithm in Algorithm 1. Prove computes a information processing function f and acquires its output out. Then it generates a proof for input and output values in commitment relation. To obtain the partial proof which includes I/O parts, it runs $\mathsf{Offline}_{\mathsf{NIZK}}$ taking a shared reference string in NIZK algorithm then gets $c_x = \prod_{i\in[1,n]} F_i^{x_i}$ where $\{x_i\}_{i\in[1,n]}$ consist of info||key. After computing a proof for I/O parts, it constructs a proof for actual function execution parts. It runs $\mathsf{Prove}_{\mathsf{NIZK}}$ by taking shared reference string for function execution $\mathsf{srs}_{\mathsf{NIZK}}$, combination of information and random value (info||key) as inputs and gets π_{NIZK} which proves correctness of function output out. Then it takes π_{XP} which proves commitment correctness from $\mathsf{Prove}_{\mathsf{XP}}$ by taking shared reference string $\mathsf{srs}_{\mathsf{XP}}$ combination of information and random value info||key. It sets π to $(c_x, \pi_{\mathsf{XP}}, \pi_{\mathsf{NIZK}})$, and returns (out, π). In Verify, Verifier takes inputs similar to the verification algorithm in Algorithm 1. To verify proofs which represent function correctness and commitment correctness, it runs verification algorithms $\mathsf{Online}_{\mathsf{NIZK}}$ and $\mathsf{Verify}_{\mathsf{XP}}$. $\mathsf{Online}_{\mathsf{NIZK}}$ takes $\mathsf{srs}_{\mathsf{NIZK}}$, a proof for I/O parts c_x, function output out, proof of the function execution π_{NIZK} as inputs. $\mathsf{Verify}_{\mathsf{XP}}$ takes $\mathsf{srs}_{\mathsf{XP}}$, commitment value h, c_x, π_{XP} which proves commitment correctness as inputs. Then Verifier takes both of verification results.

4. Security analysis

4.1. Simulation-based definition for our scheme

Proof outline: We prove Theorem 1 in section 2.5. Before describing the specific proof, we assume that the adversary cannot forge a function output and cannot distinguish a commitment in the ideal functionality (Garman et al., 2016). We will show that the adversary has an ability to forge a function output at most equal to that of an adversary in the ideal functionality and all of the distributions in real execution are computationally equal to the ideal functionality. That is, if an attack in the ideal functionality is impossible, we can ensure that the adversary in our scheme cannot forge a function output or distinguish the commitments. To show that the adversary of the ideal functionality and the real functionality have a same distribution, we first show that the adversary in the ideal functionality can be generated using an adversary in the real functionality. Then, we prove that the output of an adversary in the ideal functionality is same as the real adversary output. Finally, the simulation for an output of the real execution can be performed by assuming that distributions of the ideal world is computationally equal to distribution of the real world rather than receiving embedded values from the assumption.

In this approach, we define our system in terms of the ideal functionality implemented by a trusted party TP, which plays a role of our cryptographic protocol in the real system. In the ideal-world experiment, a collection of parties interact with the trusted party according to a specific interface. In the real-world experiment, the parties interact with each other using our protocol. We now define the experiments.

Algorithm 3 Ideal world functionality

TPsetup
  TP creates PublicTable to store h, cert
  TP creates PrivateTable to store h and info
Init(info)
  Prover sends info to TP
  if info ∈ PrivateTable **then**
    Obtain (info, h) from PrivateTable
  else
    TP generates unique random value h, and inserts (info, h) in PrivateTable
  end if
  TP returns h to Prover
Register(PublicTable, info, h)
  CA takes (info, h) from Prover
  CA sends (info, h) to TP
  if info ∈ PrivateTable ∧ h ∈ PublicTable **then**
    CA computes a certificate of h cert
  else
    abort
  end if
  CA sends (h, cert) to TP
  if cert is verified **then**
    TP stores (h, cert) in PublicTable
  else
    abort
  end if
Prove(PublicTable, h, info, f)
  if h∉ PublicTable **then** abort
  end if
  Prover obtains (h, cert) from PublicTable
  Prover runs f(info) and gets function output out
  Prover sends (f,info,out) to TP
  if info ∉ PrivateTable **then** abort
  end if
  TP gets (info, h) from PrivateTable
  if f(info) ≠ out **then** abort
  end if
  TP notifies (f, h, out) to all parties

Algorithm 3 describes the ideal world functionality where all of the entities can interact with the trusted party TP which proves correctness of the data instead of the cryptographic protocol.

TPsetup is a process where the trusted party TP sets two tables. Both tables are only allowed to append data. It is impossible to delete data and modify data in each table. While PublicTable is exposed to the public, PrivateTable is only available to TP(Trusted Party). In Init, Prover takes its private identity info as inputs. It first sends info to TP and TP checks if info is in PrivateTable. If it is in PrivateTable, TP obtains (info,h) from PrivateTable; otherwise, TP generates unique random value h and stores (info,h) in PrivateTable. After sav-

ing them, TP returns h to Prover. Prover sends (info, h) to CA.

In Register, CA takes (info, h) as inputs. Then CA sends (info, h) to CA and CA checks whether info is in the PrivateTable and h is in the PublicTable respectively. If both of them are in the table respectively, CA constructs its certificate cert of h and sends h,cert) to TP. Then, the TP checks correctness of the certificate cert and saves (h,cert) in PublicTable.

In Prove, Prover checks whether h is in PublicTable or not. If h is in PublicTable, Prover obtains (h,cert) from PublicTable. Then Prover runs f(info) and gets its output out. Prover sends function f, information info, and function output out to TP. TP checks if info is in PrivateTable, and validates if f(info) = out. If it is correct then TP notifies (f,h,out) to all parties.

We now present the process where the adversary in the ideal functionality gives output same as the adversary in the real execution. we define $\mathcal{A}'$ as an adversary which tries to forge a function output in the ideal execution. And we define $\mathcal{A}$ as an adversary which interacts with the real functionality. $\mathcal{A}'$ runs $\mathcal{A}$ internally and interacts with the ideal functionality.

When $\mathcal{A}$ outputs π_{real} where π_{real} is a proof of $\mathcal{A}$, $\mathcal{A}'$ first verifies the proof as in the real protocol. If the proof is verified, it runs the knowledge extractor on π_{real} to obtain (info, key). The simulator makes sure that (info,h) is in PrivateTable and h in PublicTable. If they are not in the tables, $\mathcal{A}'$ calls Init and Register to store info in PrivateTable, and h in PublicTable. $\mathcal{A}'$ runs Prove to obtain function output out. $\mathcal{A}'$ outputs (f, h, out) with a simulated proof π_{ideal}. Similarly, $\mathcal{A}'$ generates a simulated proof which proves correctness of commitment in Init phase using the knowledge extractor of $\mathcal{A}$ which extracts key as a witness. We now show that the output of the ideal world experiment is computationally indistinguishable from the output of the real world experiment via a series of hybrid games.

1. G_0 : This is a real execution experiment.

2. G_1 : In this game, we replace all the proofs by honest parties with simulated proofs. By Lemma 1 we show that if the proof system is computationally knowledge-sound, then $G_1 \approx G_0$.

3. G_2 : In this game, we run the knowledge extractor when encountered by the function output of any corrupted parties, and abort if the knowledge extractor fails. By Lemma 2 we show that if the proof extractor fails with negligible probability, then $G_2 \approx G_1$.

4. G_3 : In this game, we replace all the commitments in the blockchain by unique random value. By Lemma 3 we show that if commitment H is secure commitment schemes, then $G_3 \approx G_2$.

Note that G_3 represents the ideal world simulation. By summation over the previous hybrid games we show that $G_3 \approx G_0$. We conclude our proof sketch by presenting the supporting lemmas.

Lemma 1. *For all PPT adversaries $\mathcal{A}$, if NIZK exists, then the advantage of distinguishing G_0 and G_1 is $\mathcal{A}_{G1} - \mathcal{A}_{G0} \leq \epsilon$ where ϵ is the simulation failure rate.*

Proof. The simulator operates in the same manner, but we simulate proofs for honest parties. By definition, π for function output is NIZK which has efficient simulator, so we recall that the simulator will fail with at most negligible probability. Therefore ϵ is negligible.

Lemma 2. *For all PPT adversaries $\mathcal{A}$, if extractable NIZK exists, then the advantage of distinguishing G_1 and G_2 is $\mathcal{A}_{G2} - \mathcal{A}_{G1} \leq \epsilon$ where ϵ is the extraction failure rate.*

Proof. The simulator operates in the same manner, but we extract when given the output of corrupted parties. By definition, π for function output is NIZK which has a knowledge extractor. Intuitively, we can see that the extractor will fail with at most negligible probability. Therefore, our proof π has a knowledge extractor that succeed with probability $1 - \epsilon$.

Lemma 3. *For all PPT adversaries $\mathcal{A}$, the advantage of distinguishing G_2 and G_3 is $\mathcal{A}_{G3} - \mathcal{A}_{G2} \leq \epsilon$ where ϵ is negligible.*

Proof. The simulator continues to operate in the same manner as orginally described, but we now replace the commmitment h with commitments to randomly chosen value. By construction H is a statistically hiding commitment scheme and therefore the probability that an adversary can detect this substituion is negligible.

5. Experiment

In this section, we propose SIMS applications and show the performance and the key size for each application. Recall that we assume the off-chain blockchain approach in which only the commitment of the private identity is in the public blockchain.

5.1. Applications

5.1.1. Minor Check

MinorCheck function as shown in Algorithm 4 checks whether a user is minor or not without revealing the age. The function includes an "if-statement". An organization creates MinorCheck function and generates srs for NIZK for the function. After a user receives the function and srs, he/she evaluates the function with providing his/her own age and obtains an output value of True or False with NIZK proof π. The function output is True then the user is minor and the fact is proven using proof π and a commitment of the real age in a blockchain. Note that although everyone can read any content in the blockchain, no actual age information

Table 1
Experiment environment

OS	Ubuntu 16.04 LTS 64bit
CPU	Intel(R) Core(TM) i5-4670 CPU @ 3.40GHz Quad Core
Memory	DDR4 24GB

Table 2
Application informaion summary

	Minor Check	Address Check	Transcript Check	Experience Check
Number of "if-statement"	1	100	100	3
Number of inputs	1 (32 bit int)	100 (256 bit string)	100 (32 bit int)	3 (32 bit int) + 3 (2048 bit int)

is leaked but the randomized commitment is only available, which is used to validate the NIZK proof.

Algorithm 4 Minor check

```
MinorCheck(int age)
  if age < 19 then
      return True
  else
      return False
  end if
```

5.1.2. Address check

Function AddressCheck in Algorithm 5 is similar to function MinorCheck but it includes more "if-statements". The function includes 100 "if-statements". User's address which is a secret information and addresslist which contains address candidates are given. In the function, it checks whether the user address belongs to addresslist. If it is true then the address is returned; otherwise a null string is returned.

Algorithm 5 Address check

```
AddressCheck(string user_adress[],string addresslist[])
  for i in range (0,len(addresslist)) do
      if user_address[0]==addresslist[i] then
          return user_address[0]
      end if
  end for
  return ⊥
```

5.1.3. Transcript check

Another application is a transcript check system. As shown in Algorithm 6, it receives the number of subjects, scores of the subjects, and a minimum score constraint for each subject denoted as limit. In the function, it checks whether all subjects' score is no less than the given minimum score. In the evaluation, the number of subjects is given as 100. Note that compared with function AddressCheck in which only equality check is performed, function TranscriptCheck includes "greater than or equal to" operations.

Algorithm 6 Transcript check

```
TranscriptCheck(int subjectNum,int score[],int limit[])
  PassNum=0
  for i in range(0,subjectNum) do
      if score[i] ≥ limit[i] then
          PassNum++
      end if
  end for
  if PassNum==subjecNum then
      return True
  else return False
  end if
```

5.1.4. Working history check

In case of a working history check application, we assume that each company records the working history for an employee and generates its signature. The working history and the signature are transferred to an agent who uploads the data in the blockchain. Algorithm 7 shows a function which checks that every working history is valid by checking a signature for each working history in a company and all working experiences meet a given requirement. Note that we assume that all verification keys are known in the function. For a signature scheme, we adopt (Fischlin, 2003) in evaluation.

In Algorithm 7, we assume that the function receives a working history value exp_i which denotes the number of working years and its signature σ_i. We assume its signature is constructed by RSA method(Fischlin, 2003) and the signature becomes a witness, not an input value. To verify the signature it performs exponentiation according to public key vk_i. Since we assume that the signature is a witness, its verification needs a commitment operation in the proof circuit. After comparing the commitment value, it checks whether exp_i satisfies a condition such as the minimum number of

Table 3
Experiment result time

	Minor Check	Address Check	Transcript Check	Experience Check
$\mathsf{KeyGen}_{\mathsf{NIZK}}$	0.0129s	5.8479s	0.0501s	26.4016s
$\mathsf{Prove}_{\mathsf{NIZK}}$	0.0038s	1.9121s	0.0181s	9.2734s
$\mathsf{Verify}_{\mathsf{NIZK}}$	0.0011s	0.0014s	0.0011s	0.0011s
$\mathsf{Setup}_{\mathsf{XP}}$	0.0005s	3.2997s	0.0131s	0.0259s
$\mathsf{KeyGen}_{\mathsf{XP}}$	0.0031s	12.8704s	0.0528s	0.1045s
$\mathsf{Prove}_{\mathsf{XP}}$	0.0001s	0.1282s	0.0005s	0.0061s
$\mathsf{Verify}_{\mathsf{XP}}$	0.0016s	0.0016s	0.0016s	0.0016s
$\mathsf{KeyGen}_{\mathsf{SIMS}}$	0.0165s	22.0819s	0.1163s	26.5351s
$\mathsf{Prove}_{\mathsf{SIMS}}$	0.0038s	2.0402s	0.0186s	9.2795s
$\mathsf{Verify}_{\mathsf{SIMS}}$	0.0027s	0.0030s	0.0027s	0.0027s

Table 4
Experiment result size

	Minor Check	Address Check	Transcript Check	Experience Check
$\mathsf{EK}_{\mathsf{NIZK}}$	6.6805KB	13350KB	52.463KB	62919KB
$\mathsf{VK}_{\mathsf{NIZK}}$	0.2722KB	1007.1KB	4.1273KB	8.0603KB
$\mathsf{Proof}_{\mathsf{NIZK}}$	0.1244KB	0.1244KB	0.1244KB	0.1244KB
$\mathsf{PP}_{\mathsf{XP}}$	0.1245KB	804.94KB	3.2062KB	6.3501KB
$\mathsf{EK}_{\mathsf{XP}}$	0.3970KB	2616.8KB	10.415KB	20.636KB
$\mathsf{VK}_{\mathsf{XP}}$	0.3728KB	0.3728KB	0.3728KB	0.3728KB
$\mathsf{Proof}_{\mathsf{XP}}$	64B	64B	64B	64B
$\mathsf{srs}_{\mathsf{SIMS}}$	8.0270KB	17780KB	70.585KB	62955KB
$\mathsf{Proof}_{\mathsf{SIMS}}$	0.1866KB	0.1866KB	0.1866KB	0.1866KB

Algorithm 7 Working history check

ExperienceCheck(int exp_1,int_N σ_1,...,int exp_n,int_N σ_n)

```
for i in range(1,n) do
    if σ_i^{vk_i} ≠ H(exp_i) mod N then
        return False
        if exp_i ≤ exp_const_i then
            return False
        end if
    end if
end for
return True
```

working years.

5.2. Evaluation

We now describe the experiment results of the applications. We conduct four types of experiments which are a minor check, an address check, a transcript check, and a working history check. Each function consists of an "if-statement", 100 "if-statements" with equality check, 100 "if-statements" with "greater than or equal to statements", and signature verifications, respectively. We perform the experiment on an Ubuntu 16.04 LTS 64bit environment with a Intel(R) Core(TM) i5-4670 CPU @ 3.40GHz Quad Core CPU environment, DDR4 24GB memory. We adopt (Groth, 2016), and (Fiore et al., 2016) schemes for proof generation NIZK, and for a commit and prove scheme XP, respectively. Note that SIMS encompasses NIZK and XP. We utilze jsnark (Kosba) which uses libsnark (Ben-Sasson) as its submodule to generate SNARK circuits for applications.

Table 2 summarizes the function body and the input data for each application. Function MinorCheck includes a single "if-statement" and takes one 32 bit integer value as an input. Function AddressCheck consists of 100 "if-statements" with equality check and 100 256-bit string type input values. Function TranscriptCheck contains 100 "if-statements" with "greater than or equal to" statements, and takes 100 integer type input values. Function ExperienceCheck has 3 "if-statements" and exponentiation operations where $n = 3$. It takes three 32-bit integer type exp's and 2048-bit integer type σ's. Note that in the function, $vk_i = 2^{16} + 1 = 65537$ and modular $N = 2048$.

Table 3 represents the performance results for each application. Since function MinorCheck includes a single "if-statement", the performances of key generation, prove, and verify are high. Particularly, the proof time is only 3.8ms. On the other hand, the key generation and the proof time are large in function ExperienceCheck due to the signature verification. In function TranscriptCheck, the proof time denoted as $\mathsf{Prove}_{\mathsf{SIMS}}$ increases by 6 times compared with function MinorCheck.

Though functions AddressCheck and TranscriptCheck contain 100 "if-statement" and 100 inputs equivalently, the proof time and the key generation time in function AddressCheck become slow by 200 times compared with function

TranscriptCheck since the input data type size is 256 bit in function AddressCheck while it is 32 bit in function TranscriptCheck.

Table 4 presents the key (or shared reference string) size for each function. As the function becomes complicate, the evaluation key size becomes large. In addition, as input size increases, the verification key size increases. The proof size remains as constant (0.1866KB) regardless of applications.

6. Related Work

This section discusses various zk-SNARK schemes and privacy preserving blockchain approaches.

Since QSP (Quadratic Span Program) and QAP (Quadratic Arithmetic Program) have been introduced by (Johansson and Nguyen, 2013), zk-SNARK obtained constant proof size and verification. Parno et al. propose a QAP-based zk-SNARK which can be used in arbitrary functions (Parno et al., 2013) and provide a first practical implemntation of zk-SNARK. (Groth, 2016) reduces the proof size from 8 group elements to 3 group elements and the number of pairing operations needed to verify the proof from 11 to 3. The SE-SNARK scheme(Groth and Maller, 2017) establishes a concept of simulation-extractable SNARK with a similar notion to the Signatures of Knowledge (Chase and Lysyanskaya, 2006). Though it can prevent a malleability attack, it provides an efficient proof size of (Groth, 2016). As another zk-SNARK issue, zk-SNARK with updatable and universal srs (Groth et al., 2018) resolves a problem where a shared reference string requires trusted setup by allowing users to update srs independently.

By utilizing the short proof sizes and the short verification times, zk-SNARK can be used for anonymous cryptocurrencies such as (Garman et al., 2016; Kosba et al., 2016; Miers et al., 2013; Sasson et al., 2014) to conceal the transaction value. Many blockchains such as (Nakamoto et al., 2008; Wood, 2014) reveal actual transaction values in a block. On the other hand, Zcash, one of the privacy preserving cryptocurrencies (Sasson et al., 2014), enables to hide both value and address in the transaction. Using the zk-SNARK such as (Groth, 2016; Parno et al., 2013), and a commitment scheme, it is possible for a user to trust all contents of the transaction even though transactions are totally hidden. Zcash (Sasson et al., 2014) has a similarity with our proposal in that a user hides a secret value and then proves the correctness of the processed value. However, while (Sasson et al., 2014) considers only balance equation and completeness of the commitment, our approach supports various functions to process secret information. Garman et al. apply a policy to a transaction and ensure the confidentiality of the transaction concurrently (Garman et al., 2016). In addition, the scheme proves a security of the decentralized anonymous payments. In case of Zcash (Sasson et al., 2014), its security proof is based on game-based notion, on the other hand, (Garman et al., 2016) uses a ideal functionality notion. Hawk(Kosba et al., 2016) proposed constructs a privacy preserving smart contract. Although Hawk is similar to our scheme since it supports general functions, it does not cover the ID management. Hawk concentrates on the transaction case where multiple users are involved.

W3C (Sporny) (Reed) proposes an identity management scheme. It allows users to prove their identity using a verifiable claim. The user receives the verifiable claim, which proves a specific claim from a certificated agent with its signature. Then the user combines multiple verifiable claims into a verifiable presentation, which includes user's signature. The verifiable presentation is given to the verifier, and the verifier checks the verifiable presentation using a verification key in the blockchain. However, the approach in W3C has a limitation that the certificate authority should issue all possible claims in advance. In addition, it is difficult to generate an arbitrary presentation including "OR" and "NOT" operations in the approach while our SIMS allows to generate a proof for a circuit including any operation.

7. Conclusion

We propose a self-sovereign identity management system called SIMS in the blockchain environment. Using the proposed SIMS, each user has the self sovereignty to utilize personal information with preserving privacy. We define a security notion of SIMS and prove the security. Furthermore, we show various applications of SIMS such as minor check, address check, transcript check, and working history proof with experiment results including proof time, fast verification time, and key/proof size.

References

Ben-Sasson, E., . libsnark. https://github.com/scipr-lab/libsnark.

Ben-Sasson, E., Chiesa, A., Genkin, D., Kfir, S., Tromer, E., Virza, M., . libsnark (2014).

Campanelli, M., Fiore, D., Querol, A., . Legosnark: Modular design and composition of succinct zero-knowledge proofs .

Chase, M., Lysyanskaya, A., 2006. On signatures of knowledge, in: Annual International Cryptology Conference, Springer. pp. 78–96.

Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S., 2015. Geppetto: Versatile verifiable computation, in: 2015 IEEE Symposium on Security and Privacy, IEEE. pp. 253–270.

Dhillon, V., Metcalf, D., Hooper, M., 2017. The hyperledger project, in: Blockchain enabled applications. Springer, pp. 139–149.

Di Crescenzo, G., Katz, J., Ostrovsky, R., Smith, A., 2001. Efficient and non-interactive non-malleable commitment, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer. pp. 40–59.

Duffield, E., Diaz, D., 2015. Dash: A privacycentric cryptocurrency.

Eberhardt, J., Heiss, J., 2018. Off-chaining models and approaches to off-chain computations, in: Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, ACM. pp. 7–12.

Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B., 2016. Hash first, argue later: Adaptive verifiable computations on outsourced data, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 1304–1316.

Fischlin, M., 2003. The cramer-shoup strong-rsa signature scheme revisited, in: International Workshop on Public Key Cryptography, Springer. pp. 116–129.

Garman, C., Green, M., Miers, I., 2016. Accountable privacy for decentralized anonymous payments, in: International Conference on Financial Cryptography and Data Security, Springer. pp. 81–98.

Groth, J., 2010. Short pairing-based non-interactive zero-knowledge arguments, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer. pp. 321–340.

Groth, J., 2016. On the size of pairing-based non-interactive arguments, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer. pp. 305–326.

Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I., 2018. Updatable and universal common reference strings with applications to zk-snarks, in: Annual International Cryptology Conference, Springer. pp. 698–728.

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

Johansson, T., Nguyen, P.Q., 2013. Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013, Proceedings. volume 7881. Springer.

Kosba, A., . jsnark: A java library for building snarks," oblivm.com/jsnark. https://github.com/akosba/jsnark/blob/master/README.md.

Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C., 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: 2016 IEEE symposium on security and privacy (SP), IEEE. pp. 839–858.

Kosba, A., Papamanthou, C., Shi, E., 2018. xjsnark: a framework for efficient verifiable computation, in: 2018 IEEE Symposium on Security and Privacy (SP), IEEE. pp. 944–961.

Kosba, A.E., Papadopoulos, D., Papamanthou, C., Sayed, M.F., Shi, E., Triandopoulos, N., 2014. {TRUESET}: Faster verifiable set computations, in: 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 765–780.

Kshetri, N., 2017. Can blockchain strengthen the internet of things? IT professional 19, 68–72.

Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S., 2017. Concurrency and privacy with payment-channel networks, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM. pp. 455–471.

Miers, I., Garman, C., Green, M., Rubin, A.D., 2013. Zerocoin: Anonymous distributed e-cash from bitcoin, in: 2013 IEEE Symposium on Security and Privacy, IEEE. pp. 397–411.

Nakamoto, S., et al., 2008. Bitcoin: A peer-to-peer electronic cash system .

Parno, B., Howell, J., Gentry, C., Raykova, M., 2013. Pinocchio: Nearly practical verifiable computation, in: 2013 IEEE Symposium on Security and Privacy, IEEE. pp. 238–252.

Pilkington, M., 2016. 11 blockchain technology: principles and applications. Research handbook on digital transformations 225.

Poon, J., Dryja, T., 2016. The bitcoin lightning network: Scalable off-chain instant payments.

Rackoff, C., Simon, D.R., 1991. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack, in: Annual International Cryptology Conference, Springer. pp. 433–444.

Reed, D., . decentralizedIdentifier. https://w3c-ccg.github.io/did-spec/.

Rouse, M., 2019. self-sovereign identity. https://searchsecurity.techtarget.com/definition/self-sovereign-identity.

Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M., 2014. Zerocash: Decentralized anonymous payments from bitcoin, in: 2014 IEEE Symposium on Security and Privacy, IEEE. pp. 459–474.

Sporny, M., . verifiable credential. https://www.w3.org/TR/2019/CR-vc-data-model-20190725/#zero-knowledge-proofs.

Stokkink, Q., Pouwelse, J., 2018. Deployment of a blockchain-based self-sovereign identity, 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), IEEE. pp. 1336–1342.

Tobin, A., Reed, D., 2016. The inevitable rise of self-sovereign identity. The Sovrin Foundation 29.

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

Zyskind, G., Nathan, O., et al., 2015. Decentralizing privacy: Using blockchain to protect personal data, in: 2015 IEEE Security and Privacy Workshops, IEEE. pp. 180–184.