Privacy-preserving Identity Management System

Jeonghyuk Lee, Jaekyung Choi, Jihye Kim, and Hyunok Oh

[1] Hanyang University, Seoul, Korea
[2] Kookmin University, Seoul, Korea
ahoo791@hanyang.ac.kr cjk2889@kookmin.ac.kr jihyek@kookmin.ac.kr
hoh@hanyang.ac.kr

Abstract. Recently, a self-sovereign identity model has been researched actively as an alternative to the existing identity models such as a centralized identity model, federated identity model, and user-centric model. The self-sovereign identity model allows a user to have complete control of his identity. Meanwhile, the core component of the self-sovereign identity model is data minimization. The data minimization signifies that the extent of the exposure of user private identity should be minimized. As a solution to data minimization, zero-knowledge proofs can be grafted to the self-sovereign identity model. Specifically, zero-knowledge Succinct Non-interactive ARgument of Knowledges(zk-SNARKs) enables proving the truth of the statement on an arbitrary relation. In this paper, we propose a privacy-preserving self-sovereign identity model based on zk-SNARKs to allow any type of data minimization beyond the selective disclosure and range proof. The security of proposed model is formally proven under the security of the zero-knowledge proof and the unforgeability of the signature in the random oracle model. Furthermore, we optimize the proving time by checking the correctness of the commitment outside of the proof relation for practical use. The resulting scheme improves proving time for hash computation (to verify a commitment input) from 0.5 s to about 0.1 ms on a 32-bit input.

Keywords: Identity model · Self-sovereign identity model · Verifiable credential data model · DIDs · zero-knowledge proof · zk-SNARKs · Commit-and-Prove

1 Introduction

For decades, how an individual's identity can be trusted in a digital world has been treated as an important issue. In the early days, a centralized identity model where centralized authority is given a control of digital identity has been used for years. In this model, a user identity is managed and authenticated in a single centralized authority that is combined with the service provider. However, the user identity for accessing the service is completely subordinated to the centralized authority, thus it is difficult for the user identity to be used interoperably and the user identity can be vulnerable if the centralized authority is compromised. A federated identity model where the identity provider is separated from the service provider and multiple service providers establish a kind

of trust relationship in one federated group was proposed as an alternative to the centralized model. If a user accesses one service provider by providing authenticated user identity from the trusted identity provider, the user identity can be shared with other services in the federated group. Despite the improved interoperability, trust between the services in a federated group is required, and the service provider should fully trust the identity provider. So, the extent of services that can be accessed from a single identity provider is limited to the federated group. The user-centric model is proposed to enhance further the interoperability of identity. In a user-centric model, a user can choose the identity providers freely and utilize them to access any services; the trust relationship between services is not required. However, all existing identity models have deficiencies in terms of control because the permission to manage and authenticate a user identity is completely subordinated to the identity provider [2]. Since user's identity is stored in the identity provider's server, the identity can be invalid if the server is compromised or the identity service ends.

Meanwhile, a self-sovereign identity model that enhances a control of user identity without missing the interoperability has received attention as an alternative to the existing identity models after the rise of the blockchain such as bitcoin [32] and ethereum [45]. Since the blockchain allows the identity management to be fully decentralized and guarantees identity persistence of with the non-malleability of the identity, a user can control his/her own identity without the identity provider. Recently, studies that establish the self-sovereign identity model using the blockchain have been proposed [3, 31, 44]. Specifically, the World Wide Web Consortium(W3C) stipulates a verifiable credential data model [12] that uses decentralized identifiers(DIDs) structure [38] as a building block. In the verifiable credential data model, an issuing organization like a government issues a verifiable credential that certifies claim data about user identity. Multiple verifiable credentials are reorganized as a verifiable presentation by the user in accordance with the request of the verifier. The verifier accesses user's DIDs, which includes data for the credential verification stored in the blockchain and efficiently verifies the verifiable presentation.

The verifiable credential data model enables a user to utilize his registered identity permanently and it helps the user control of his/her identity. The verifiable credential data model has advantages in terms of persistence and control, but has limitations in that the user's privacy is not protected to the maximum extent. Since the verifiable presentation is composed of the verifiable credentials that are not an encrypted form or a committed form, the verifier can access the claim information in the verifiable credentials. For instance, when the verifier only needs to know whether the user is a minor or not, the verifier can recognize user's exact age from the verifiable credentials. Thus, several research groups [44, 42] including the standard group [12] tried to explore zero-knowledge proof to the verifiable credential data model. Zero-knowledge proof enables a user to prove a claim/statement without revealing the knowledge of the claim. They attempt to supplement the privacy in the existing verifiable credential data model using the notion. Specifically, their works support a selective disclosure of claim [12,

44, 42] and range proof of claim [42] using the CL signature scheme [9] and the range proof scheme [36].

However, if complex computations rather than selective disclosure or the range check are required for the verifiable presentation generation, the above proof schemes [9, 36] cannot be utilized as building blocks in the verifiable presentation generation. For instance, if the user should prove his/her financial asset level, the components of the assets such as goods and debt should be evaluated by a complex appraisal function. Since the selective disclosure and range check function cannot express the appraisal function in some cases, it is not enough to construct the verifiable credential data model based on CL signatures and range proof schemes. As an alternative of CL signature based scheme, the zero-

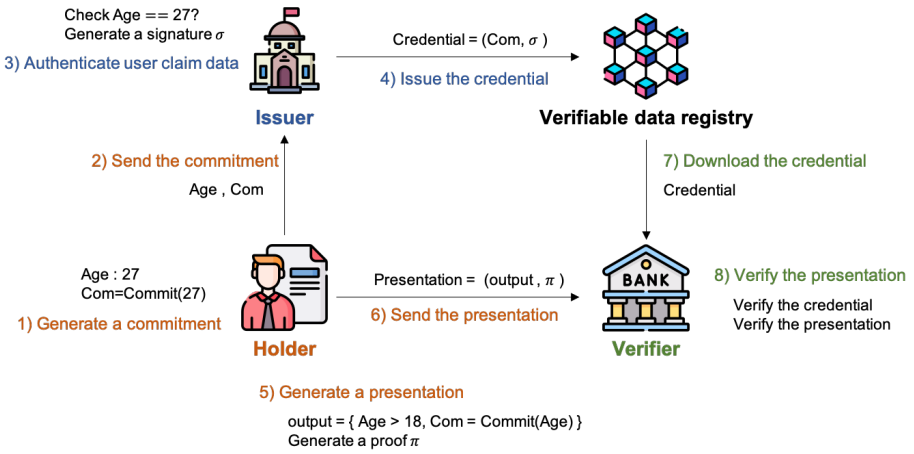

Fig. 1: Basic structure of our proposal

knowledge Succinct Non-interactive ARgument of Knowledges(zk-SNARKs) [19, 7, 25, 29, 15, 35, 20] can be used as building blocks for the data minimization. zk-SNARKs is capable of proving the correctness of arbitrary statements and helps the user to prove the validity of the verifiable presentation that is composed of the complex computations with zero-knowledge. In fact, Yang and Li utilized the zk-SNARKs for the data minimization recently [47]. In their construction, the verifiable credential is issued as a committed form with its zk-SNARKs proof, and the user also generates the verifiable presentation that expresses the correctness of the credential and its one-time use. Additionally,in their work, the zk-SNARKs is utilized to prove the possession of a specific attribute that is issued in advance. Furthermore, since they utilize zk-SNARKs as only a way of improving verification efficiency, their work confines the presentation format as the form of showing only the possession of attribute despite the expression power of the zk-SNARKs. As stated above, the function that requires complex computation such as credit appraisal and the qualification for tax exemption

cannot be expressed through only proving the possession of attributes. Moreover, their works require the issuer to generate zk-SNARKs proof for all user claims, which is undesirable for the availability. Thus, we propose a new zero-knowledge verifiable credential data model to focus on supporting complicated verifiable presentation.

In our scheme, the trusted issuer issues the verifiable credential as a committed form with its signature, and the user presents the verifiable presentation that only discloses the minimum information derived by computing the user claim data to the verifier. The verifiable credential is included in the verifiable presentation directly in other works [44, 42]. In contrast, our approach assumes that the verifiable presentation is derived from the user claim data and the verifier verifies connectivity of the given verifiable presentation and the verifiable credential that is recorded in advance. Fig. 1 describes a flow of our zero-knowledge verifiable credential data model.

As shown in Fig. 1, our zero-knowledge verifiable credential data model assumes triple entities such as an issuer, a holder and a verifier. A holder who has private user claim data, such as subject's age, commits the user claim data. The holder then sends the private value and its commitment to the issuer. When the issuer receives the values, the issuer authenticates the private value and checks the correctness of the commitment. If the values are valid, then the issuer signs on the commitment and issues the commitment and its signature as a verifiable credential. The verifiable credential is recorded in the verifiable data registry, and the holder keeps the address of the verifiable credential in his DID documents. When the holder is required to present his qualification by the verifier, the holder generates a minimized public user claim data from his private user claim data directly; and guarantees the correctness of the public user claim data by presenting it with the zk-SNARKs proof. The verifier verifies if the public user claim data is derived from the private user claim data that is registered as a committed form by verifying the zk-SNARKs proof. When the verifier verifies the proof, the verifiable credential that is used as a input of a statement is downloaded from the verifiable data registry.

We achieve a versatility of the verifiable presentation with privacy by utilizing zk-SNARKs. However, it can cause inefficiency with respect to the proving time if an appropriate computation circuit size is not considered. Since the proving time of zk-SNARKs increases proportionally to the circuit size, reducing the circuit size as little as possible is desirable. In the scheme with committed credentials, the computation circuit should include a process of the commitment correctness check. A single commitment check process increases the circuit size by as much as about 25,000 constraints if, for example, SHA-256 hash algorithm is used as the commitment scheme [39]. Thus, we remove the commitment check process from the computation circuit using Commit-and-Prove zk-SNARKs(CP-SNARKs) [14, 11]. CP-SNARKs allows the verifier to verify the commitment correctness apart from the circuit; the circuit checks only the equality of the commitment in the circuit; while verifying the commitment outside the circuit.

Table 1: Comparison of self sovereign identity models

	Ours	Sovrin [44]	IPv8 [42]	YL20 [47]	Uport [31]
Control	O	O	O	O	O
Data minimization	O	O	O	O	X
Expression power	Unlimited	Selective disclosure	Selective & Range disclosure	Selective disclosure	-
Persistence	O	O	O	O	O
ZKP type	zk-SNARKs [11]	CL02 [9]	CL02 [9], PB10[36]	zk-SNARKs [35, 19, 20]	-
ZKP optimization	O	-	-	X	-

Control : The capability of controlling the identity by the user himself
Data minimization : The minimization of the user identity disclosed to the verifier
Expression power : The flexibility of the form of the processed identity given the verifier
Persistence : The persistence of the issued identity in the repository
ZKP type : The used zero-knowledge proof scheme in the construction
ZKP optimization : The optimization of zero-knowledge proofs in terms of the proving time.

1.1 Our contribution

Model We define a privacy-preserving verifiable credential data model following the verifiable credential data model standard [12]. Our credential data model improves data minimization in the standard model. The credential model is equipped with the formal security notion and the model can be applied to other standard-based identity models. As shown in Table 1, other privacy-preserving models support only limited relations such as selective disclosure or range disclosure. In contrast, our model has an unlimited expression power in terms of the verifiable presentation. Notwithstanding the high flexibility of the expression, our model supports a control and the persistence of the user identity in common with the other models.

Construction We provide a specific construction of the privacy-preserving model via zk-SNARKs. Our construction allows any type of zk-SNARKs such as pairing based pre-processing zk-SNARKs [19, 20, 25] and the universal reference string based zk-SNARKs [7, 29, 15]. Since we supports the general construction of the verifiable credential data model, diverse zk-SNARK libraries such as libsnark [5], snarkjs [4] can be applied to the construction implementation.

Security Though many identity models have been proposed [44, 31, 42, 47], to the best of our knowedlge, the formal security proof is not provided in most cases. We provide the formal security proof of the proposed model. Our security notion relies on the simulation-based approach [16]. The simulation-based approach assumes the ideal credential data model where a trusted party substitutes the cryptographic protocol in the proposed model. We show that the advantages of the proposed model are converged into the advantages of the ideal credential data model by simulating an adversary of the ideal credential data model from

our credential data model. Additionally, the security of our construction implies the resistance of the replay attack.

Optimization We optimize the construction to reduce proving time. Particularly, the commitment correctness is checked outside the proof circuit and only equality of I/O is checked in the proof circuit by formatting the commitment as I/O of the proof. If the commitment check process can be removed from the proof circuit, it can improve the efficiency in terms of the proving time notably. Specifically, the proving time takes 3.8 ms in the case of a single if-statement on a 32-bit single input.

1.2 Organization

We describe related works in Section 2. In Section 3, we explain background of our design. We describe definition of the verifiable credential data model and our zero-knowledge verifiable credential data model in Section 4. We demonstrate specific construction of our signature scheme in Section 5. Section 6 presents the security proofs of all constructions and Section 7 analyzes the experiment results of our scheme. We draw a conclusion in Section 8.

2 Related work

2.1 Existing self-sovereign identity models

The self-sovereign identity model is first introduced by Allen [2]. In [2], the notion of self-sovereign identity model and its properties were defined. The self-sovereign identity model has been researched actively by various research groups including the W3C group. W3C defines a standard of the self-sovereign identity model called verifiable credential data model [12] and DIDs notion [38]. After the standards were founded, there has been an active research involving the self-sovereign identity model [21, 30, 13, 48]. Naik and Jenkins [31] design uPort that is a self-sovereign identity model based on the ethereum and InterPlanetary File Systems(IPFS). Kassem et al. [3] propose a design similar to uPort. The design uses both permissionless blockchain and permissioned blockchain as building blocks, whereas, uPort utilizes only permissionless blockchain. Sovrin [44] is an open source project that is operated on the hyperledger indy. Sovrin works in accordance with the verifiable credential data model, and DIDs are managed in the permissioned blockchain. The verifiable credential data model is operated by the smart contract on the permissioned blockchain. Stokkink and Pouwelse [43] propose a model based on the personalized blockchain like TrustChain [34] and the Tangle [37]. Additionally, several researches[1, 33, 41] apply the verifiable credential data model and DIDs structures to other applications such as eID [1], the biometric authentication [33, 22] and finance applications [41]. Recently, zero-knowledge proof [10](ZKP) was grafted onto the standard model for data minimization [44, 42, 22, 47]. Though ZKP allows data minimization, the expression

of the minimized data is limited unless zk-SNARKs [35, 19, 20, 25, 15, 11, 29, 7] is utilized as building blocks of the construction. Only Yang and Li utilize zk-SNARKs as a building block of the construction, however, they limited the expression to the ownership of the identity despite of the flexible expression of zk-SNARKs. Furthermore, their approach is inefficient in terms of the proving time since they encode the commitment circuit into the proof circuit.

2.2 zk-SNARKs

This section discusses various zk-SNARKs schemes. Since Gennaro et al. [17] have proposed first zk-SNARKs notion where arbitrary expressions are supported using Quadratic Span Program(QSP) or Quadratic Arithmetic Program(QAP) notion [23], several practical zk-SNARKs schemes have been proposed [35, 19, 20, 25]. Parno et al. [35] provided the first practical implementation of zk-SNARKs that can support arbitrary expression. Groth [19] reduces the proof size from 8 to 3 group elements and the number of pairing operations from 11 to 3 in verification. Groth and Maller [20] established a simulation extractability notion, which prevents the adversary from forging a proof even if the adversary can access the simulated proof. In the protocol, the proof size remains as those of Gro16 [19]. However, the simulation extractable zk-SNARKs is based on SAP circuit, which causes proving time inefficiency because the size of the Square Arithmetic Program(SAP) circuit is twice as much as that of the QAP circuit. Several works have attempted to construct the SE-SNARKs scheme based on the QAP circuit [6, 28] and Kim et.al [25] accomplished the QAP based SE-SNARKs while remaining the proof size as three.

However, several researches [46, 40, 7, 29, 15, 8] construct zk-SNARKs scheme without using QAP or QSP expression because pairing based zk-SNARKs requires a trusted setup. GKR protocol [18] obtains a sublinear proof based on the sum-check protocol. However, the protocol does not support constant verification. Thus, there have been several studies [46, 40] based on the GKR protocol to improve the efficiency. Several studies proposeed the universal setup zk-SNARKs based on the polynomial commitment scheme. Sonic [29] and Plonk [15] accomplished the universal string-based zk-SNARKs that supports constant verification using Kate polynomial commitment scheme [24] as building blocks. Furthermore, Bulletproof [7] proposes an inner-product argument scheme that has a logarithmic proof size with the untrusted setup. Bunz et al. [8] propose a polynomial commitment scheme that does not require the trusted setup using the unknown order group.

3 Preliminaries

3.1 Notation

We write $y \leftarrow x$ for substitution x on y. We write $y \leftarrow S$ for sampling y from S if S is a set. We write $y \leftarrow A(x)$ for a probabilistic algorithm on input x returning

output y. When a probabilistic algorithm $A(x)$ has a private input r, we denote $A(x;r)$. We state $f(\lambda)$ is *negligible* if $f(\lambda) \approx 0$. We denote a concatenation as $||$. Given a scheme Π, its all operations are denoted by $\Pi.\mathsf{name}$. We denote by $a_{j,i}$ the element of $l \times N$ matrix $\mathbf{A}$ and we denote by a_i the element of l length vector $\vec{a}$. Let $\mathcal{R}$ be a relation generator that given a security parameter λ in unary returns a polynomial time decidable relation $R \leftarrow \mathcal{R}(1^\lambda)$. We denote $\mathcal{R}_\lambda$ as the set of relations that $\mathcal{R}(1^\lambda)$ outputs. We call ϕ the instance and w the witness for $(\phi, w) \in R$. We denote all of $\mathcal{A}$'s inputs and outputs for an algorithm $\mathcal{A}$ by $trans_{\mathcal{A}}$.

3.2 Bilinear Groups

We recall the definition of bilinear groups [25]. A Bilinear group generator $\mathcal{BG}$ takes as input a security parameter and outputs a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$. $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order p with generator $G \in \mathbb{G}_1$, $H \in \mathbb{G}_2$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerative bilinear map (i.e $e(G^a, H^b) = e(G, H)^{ab}$ and $e(G, H)$ generates $\mathbb{G}_T$).

3.3 Commitment Schemes

Definition 1. *A commitment scheme is a tuple of algorithms* $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ *as follows.*

- Setup *is a PPT setup algorithm that takes as input a security parameter and outputs a commitment key ck. The commitment key ck includes the description of input space $\mathcal{D}$, commitment space $\mathcal{C}$, opening space $\mathcal{O}$.*
- Commit *is a PPT algorithm that takes as input a commitment key ck, a value u and outputs a commitment c and an opening o.*
- VerCommit *is a deterministic polynomial time algorithm that takes as input a commitment key ck, a commitment c, a value u, and an opening o, and returns 0(reject) or 1(accept).*

A commitment scheme Com *satisfies the notion of correctness, binding and hiding [11].*

3.4 Digital signatures

Definition 2. *A digital signature is a set of algorithms* $\mathsf{Sig} = (\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$ *where*

- Keygen *takes as input a security parameter λ and returns a key pair sk, vk the signing key and the verification key.*
- Sign *takes as input a message m, the secret key sk and returns σ.*
- Verify *takes as input a message m, the verification key vk, the signature σ and returns 1 if the σ is valid signature or 0, otherwise.*

A digital signature scheme Sig *satisfies a notion of unforgeability described below.*

Unforgeability Sig satisfies unforgeability if $\mathbf{Adv}^{unforge}_{\mathsf{Sig},\mathcal{F}}(\lambda) \approx 0$ for any PPT adversary $\mathcal{F}$ where the execution time is at most t and the number of signing queries is at most q_{sig}. An adversary who wants to succeed a valid signature forgery executes chosen message attack cma until the number of signing queries amounts to q_{sig}. The adversary succeeds a valid forgery if the adversary generates a signature of a new message. Formally we define $\mathbf{Adv}^{unforge}_{\mathsf{Sig},\mathcal{F}}(\lambda) = \Pr[\mathcal{G}^{unforge}_{\mathsf{Sig},\mathcal{F}}(\lambda)]$ where the game $\mathcal{G}^{unforge}_{\mathsf{Sig},\mathcal{F}}$ is defined in Algorithm 1.

Algorithm 1 Unforgeability game $\mathcal{G}^{unforge}_{\mathsf{Sig},\mathcal{F}}(\lambda)$

$\mathcal{G}^{unforge}_{\mathsf{Sig},\mathcal{F}}(\lambda)$
  $(sk, vk) \leftarrow \mathsf{Keygen}(\lambda)$
  repeat
    $\mathsf{forge} \leftarrow \mathcal{F}^{\mathsf{Sign}(\cdot)}(\mathsf{cma}, vk)$
  until the number of queries is q_{sig}
  $(m^*, \sigma) \leftarrow \mathcal{F}(\mathsf{forge})$
  if $\mathsf{Verify}(m^*, vk, \sigma) = 1$ and m^* was not queried of $\mathsf{Sign}(\cdot)$ **then**
    return 1 **else** return 0
  end if

3.5 Simulation-extractable zk-SNARK

We recall the definition of simulation-extractable zk-SNARK [20, 25, 27]

Definition 3. *A zero-knowledge succinct non-interactive arguments of knowledge(zk-SNARK) for $\mathcal{R}$ is a set of quadruple algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimProve})$ as follows.*

- Setup *is a PPT setup algorithm that takes as input a relation $R \in \mathcal{R}_\lambda$ and returns a common reference string crs and a simulation trapdoor τ.*
- Prove *is a PPT algorithm that takes as input a common reference string crs, an instance ϕ and a witness w for $(\phi, w) \in R$, and returns a proof π.*
- Verify *is a deterministic polynomial time algorithm which takes as input a common reference string crs, an instance ϕ and a proof π, and returns 0(reject) or 1(accept).*
- $\mathsf{SimProve}$ *is a PPT algorithm which takes as input a common reference string crs, a simulation trapdoor τ, and an instance ϕ. The algorithm returns a simulated proof π.*

zk-SNARK Π satisfies completeness, knowledge soundness, zero-knowledge, and succinctness as described below.

Perfect completeness : Perfect completeness stipulates that a prover with a witness who is given a true statement can convince a verifier.

For all $\lambda \in \mathbb{N}$, for all $R \in \mathcal{R}_\lambda$ and for all $(\phi, w) \in R$:

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

Computational soundness : Computational knowledge soundness states that the prover must know a witness and the witness should be extracted efficiently from a knowledge extractor. Proof of knowledge requests every adversarial prover $\mathcal{A}$ to generate an accepting proof, there must be an extractor $\chi_\mathcal{A}$ which outputs a valid witness taking a same input of $\mathcal{A}$. Formally, we define $\mathbf{Adv}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) = \Pr[\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)]$ where $\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}$ is defined as follows.

Algorithm 2 Knowledge soundness game $\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}$

$\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)$
  $R \leftarrow \mathcal{R}(1^\lambda)$
  $(crs, \tau) \leftarrow \mathsf{Setup}(R)$
  $(\phi, \pi) \leftarrow \mathcal{A}(crs)$
  $w \leftarrow \chi_\mathcal{A}(trans_\mathcal{A})$
  $\mathcal{A}$ wins if $\mathsf{Verify}(crs, \phi, \pi) = 1$ and $(\phi, w) \notin R$ and fails otherwise.

An argument system Arg is considered computationally sound if for any PPT adversary adversary $\mathcal{A}$, there exists a PPT extractor $\chi_\mathcal{A}$ where $\mathbf{Adv}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) \approx 0$

Perfect zero-knowledge : Perfect zero-knowledge stipulates that a proof does not disclose any information about the witness besides the truth of the instance. The statement is certified by a simulator which cannot access a witness but has some trapdoor information that allows simulating proofs. Formally, we define $\mathbf{Adv}^{zk}_{Arg,\mathcal{A}}(\lambda) = 2\Pr[\mathcal{G}^{zk}_{Arg,\mathcal{A}}(\lambda)] - 1$ such that the game $\mathcal{G}^{zk}_{Arg,\mathcal{A}}$ is defined as follows.

The argument system is considered perfect zero-knowledge if $\mathbf{Adv}^{zk}_{Arg,\mathcal{A}}(\lambda) = 0$ for all PPT adversaries $\mathcal{A}$.

Simulation-Extractability : Simulation-Extractability stipulates that any adversary $\mathcal{A}$ who can access a simulated proof for a false instance cannot forge the proof to another proof for a false instance. Formally, we define $\mathbf{Adv}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) = \Pr[\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)]$ where the game $\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}$ is defined as follows.

zk-SNARK is considered simulation extractable if there exists an extractor $\chi_\mathcal{A}$, for any PPT adversary $\mathcal{A}$, where $\mathbf{Adv}^{proof-ext}_{Arg,\chi_\mathcal{A},\mathcal{A}}(\lambda) \approx 0$.

Algorithm 3 Zero-knowledge game $\mathcal{G}^{zk}_{Arg,\mathcal{A}}$

$\mathcal{G}^{zk}_{Arg,\mathcal{A}}(\lambda)$
  $R \leftarrow \mathcal{R}(1^\lambda)$
  $(crs, \tau) \leftarrow \mathsf{Setup}(R)$
  $b \leftarrow \{0, 1\}$
  if $b = 0$ **then**
    $P^b_{crs,\tau}(\phi_i, w_i)$ returns π_i where $\pi_i \leftarrow \mathsf{Prove}(crs, \phi, w)$ and $(\phi_i, w_i) \in R$
  else
    $P^b_{crs,\tau}(\phi_i, w_i)$ returns π_i where $\pi_i \leftarrow \mathsf{SimProve}(crs, \phi, \tau)$ and $(\phi_i, w_i) \in R$
  end if
  $b' \leftarrow \mathcal{A}^{P^b_{crs,\tau}(\phi_i, w_i)}$
  $\mathcal{A}$ wins if $b = b'$ and fails otherwise.

Algorithm 4 Simulation extractable knowledge soundness game $\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}$

$\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)$
  $R \leftarrow \mathcal{R}(1^\lambda)$, $Q \leftarrow 0$
  $(crs, \tau) \leftarrow \mathsf{Setup}(R)$
  repeat
    $\pi_i \leftarrow \mathsf{SimProve}(crs, \tau, \phi_i)$
    $Q \leftarrow Q \cup \{\phi_i, \pi_i\}$
  until $(\phi, \pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}_{crs,\tau}}(crs)$
  $w \leftarrow \chi_\mathcal{A}(trans_\mathcal{A})$
  $\mathcal{A}$ wins if $\mathsf{Verify}(crs, \phi, \pi) = 1$, $(\phi, w) \notin R$ and $(\phi, \pi) \notin Q$ and fails otherwise.

3.6 Commit and Prove zk-SNARK

We recall the definition of Commit and Prove zk-SNARKs [11].

Definition 4. *Let $\mathcal{R}_\lambda$ be a set of relations R over $\mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_w$ such that $\mathcal{D}_u$ splits over l arbitrary domains ($\mathcal{D}_1 \times \cdots \times \mathcal{D}_l$) for some arity parameter $l \geq 1$. Let $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ be a commitment scheme(definition 1) whose input space $\mathcal{D}$ is such that $\mathcal{D}_i \subset \mathcal{D}$ for all $i \in [l]$. A commit and prove zk-SNARK for Com and $\mathcal{R}_\lambda$ is zk-SNARK for a set of relations $\{\mathcal{R}_\lambda^{\mathsf{CP}}\}$ such that:*

- *all $R_{\mathsf{CP}} \in \mathcal{R}^{\mathsf{CP}}$ is represented by a pair (ck, R) where $ck \in \mathsf{Com.Setup}$ and $R \in R_\lambda$.*
- *R_{CP} is over pairs $(\phi_{\mathsf{CP}}, w_{\mathsf{CP}})$ where the statement is $\phi_{\mathsf{CP}} = (\phi, \vec{c}) \in \mathcal{D}_\phi \times \mathcal{C}^l$, the witness is $w_{CP} = (\vec{u}, \vec{o}, w) \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_l \times \mathcal{O}^l \times \mathcal{D}_w$, and the relation R_{CP} holds iff*
  $\bigwedge \mathsf{VerCommit}(ck, c_j, u_j, o_j) = 1 \wedge R(\phi, \vec{u}, w) = 1$

Definition 5. *A CP-SNARKs is a triple of algorithms $\mathsf{CP} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as follows.*

- Setup *takes as input a pair $R_{\mathsf{CP}} = (R, ck)$ and outputs the common reference string crs.*
- Prove *takes as input a common reference string crs, an instance $\phi_{\mathsf{CP}} = (\vec{c}, \phi)$ and witness $w_{\mathsf{CP}} = ((\vec{u}, \vec{o}), w)$ and outputs a proof π_{CP}.*
- Verify *takes a common reference string crs, an instance ϕ_{CP}, and a proof π_{CP} and outputs 0(reject) or 1(accept).*

The CP-SNARKs satisfies perfect completeness, computational knowledge-soundness, and perfect zero-knowledge [11].

4 Zero-knowledge verifiable credential data model

4.1 Verifiable credential data model

W3C establishes a verifiable credential data model that is an open standard for digital credential [12]. The verifiable credential data model stipulates a specific way of utilizing a verifiable credential and a verifiable presentation that is composed of the verifiable credential. Using the verifiable presentation and DIDs structure, the credential data of an individual can be verified via the verifiable data registry. We first recall the components of the verifiable credential data model and describe how the verifiable credential data model works [12].

Entities : The verifiable credential data model supposes quadruple of entities Issuer, Holder, $\mathsf{Subject}$ and $\mathsf{Verifier}$ who join the protocol as follows.

- Issuer is a role that performs by asserting claims about one or more subjects, creating a verifiable credential from these claims, and transmitting the verifiable credential to a holder.

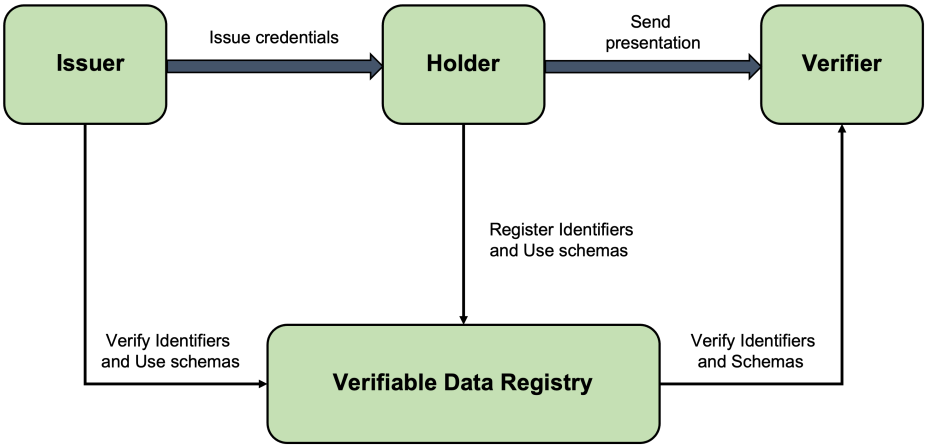

Fig. 2: Verifiable credential data model

- Holder is a role an entity might perform by possessing one or more verifiable credentials and generating verifiable presentations from them
- Subject is an entity about which claims are made. The entity can include human being, animals, and things.
- Verifier is a role an entity performs by receiving one or more verifiable credentials(optionally inside a verifiable presentation) for processing.

In many cases, Holder of the verifiable credential is Subject. However, the permission of Subject to generate the verifiable presentation can be delegated to different Holder. For instance, a parent might hold the verifiable credentials of a child, or a pet owner might hold the verifiable credentials of their pet.

Verifiable credential : A credential is a set of claims made by an issuer about the subject. A verifiable credential is a kind of credential with authorship that can be cryptographically verified, making the credential tamper-evident. An issuer issues the verifiable credential, and the issuer is assumed to be trusted in the model.

Verifiable presentation : A verifiable presentation is data derived from one or more verifiable credentials, issued by one or more issuers, that is shared with a specific verifier. The verifiable presentation is also a tamper-evident presentation encoded in such a way that the authorship of the data can be trusted after a process of cryptographic verification. The verifiable presentation is generated by a holder.

Verifiable data registry : A verifiable data registry is role a system might perform by mediating the creation and verification of identifiers, keys and other

relevant data, such as verifiable credential schemas, revocation registries, issuer public keys that might be required to use verifiable credentials. Examples of the verifiable data registries are decentralized ledger, a trusted database, and decentralized database.

Figure 2 represents how the verifiable credential data model works. When Issuer issues a verifiable credential of Subject, Holder composes verifiable credentials into one verifiable presentation(the verifiable credential can be submitted as itself) and sends the verifiable presentation to Verifier. Verifier check the correctness of the verifiable presentation referring to data such as DIDs that is stored in the verifiable data registry.

4.2 Zero-knowledge verifiable credential data model

We now describe how the privacy preserving verifiable credential data model works. In the privacy-preserving model, the process is split into three phases that are zero-knowledge claim phase zkCL, zero-knowledge verifiable credential phase zkVC and zero-knowledge verifiable presentation phase zkVP. In zkCL phase, Holder generates a committed claim of the Subject and Issuer verifies the correctness of the claim. In zkVC, Issuer generates privacy preserved verifiable credential based on the given committed claims. The verifiable credential in the existing model should be kept only in the Holder's wallet, however, the privacy-preserving verifiable credential can be included in the DIDs. zkVP treats all the process of verifiable presentation. Holder presents the zero-knowledge verifiable presentation to Verifier and Verifier checks whether the presentation data is correct or not referring to DIDs. All the phases consist of Setup, Gen, Verify respectively. Specifically, the phases are as follows.

Definition 6. *A zero-knowledge claim phase* zkCL *is a set of algorithms* (Setup, Gen, Verify) *as follows.*

- Setup *takes as input a security parameter λ, a maximum number of attributes in one claim N and outputs a public parameter pp.*
- Gen *takes as input a public parameter pp, an attribute set $\vec{u_j}$ and outputs a verifiable claim* claim.
- Verify *takes as input a public parameter pp and verifiable claim* claim *and outputs 0(reject) or 1(accept).*

Definition 7. *A zero-knowledge verifiable credential phase* zkVC *is a set of algorithms* (Setup, Gen, Verify) *as follows.*

- Setup *takes as input a security parameter λ and outputs an issuer signing key-verification key pair (sk_j, vk_j).*
- Gen *takes as input a verifiable claim* claim *and issuer signing key sk_j and outputs the zero-knowledge verifiable credential* cred_j *and it is uploaded to the verifiable data registry.*
- Verify *takes as input a signature verification key vk_j and the verifiable credential* cred_j *and outputs 0(reject) or 1(accept).*

Definition 8. *A zero-knowledge verifiable presentation phase* zkVP *is a set of algorithms* (Setup, Gen, Verify) *as follows.*

- Setup *takes as input a security parameter* λ *and a public parameter pp, a relation for the presentation* R_k^* *and outputs corresponding common reference string* crs_k.
- Gen *takes as input a common reference string* crs_k*, zero-knowledge credential set* $\vec{\text{cred}}$*, a presentation function input in and an attribute value matrix* $\boldsymbol{u}$*, a secret value set of* Holder $\vec{o}$ *and outputs a privacy-preserving verifiable presentation* pres.
- Verify *takes as input a common reference string* crs_k*, a zero-knowledge verifiable presentation* pres*, the verifiable credential set* $\vec{\text{cred}}$*, the issuer verification key set* $\vec{vk}$ *and outputs 0(reject) or 1(accept).*

We assume that zkVP.Setup is performed by a certificated authority. All claim data originated from Holder, and Issuer is given power only to check the correctness of the claim and register the committed claim as a form of verifiable credential. In our protocol, though Issuer has permission to identify actual claim data, an authority of claim to use is limited to only Holder. We call our privacy-preserving verifiable credential data model(zkCL, zkVC, zkVP) a privacy-preserving identity management systems PIMS.

Security model We define a security model of the privacy preserving self-sovereign identity management scheme. The security model follows the simulation based definition [16]. We assume an ideal world execution where all the cryptographic constructions are substituted to a trusted party TP, and an adversary cannot forge a function output and distinguish messages of two commitments. We suppose 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. We define the ideal world execution and the real world execution formally as follows.

Definition 9. ***Ideal world execution*** $\text{Ideal}_{\text{TP},\mathcal{S},\Sigma}$*: The ideal world attacker* $\mathcal{S}$ *selects several subset of the* $P_1, \ldots, P_n$ *parties to corrupt and informs the* TP*. The attacker controls parties to behave arbitrarily, the honest ones follow a set of strategies* Σ*. All parties interact via messages among each subjects implementing the ideal world execution described in Algorithm 9,10,11.*

Definition 10. ***Real world execution*** $\text{Real}_{\text{PIMS},\mathcal{A},\Sigma}$ *: The real world attacker* $\mathcal{A}$ *controls a subset of the parties* $P_1, \ldots, P_n$ *interacting with the our privacy-preserving verifiable credential data model. The honest parties execute the commands output by the strategy* Σ *using the* PIMS *while the attacker can control parties to behave arbitrarily. We assume that all parties can interact with a verifiable data registry.*

Definition 11. *We say that real functionality* PIMS *securely emulates the ideal world execution* $\text{Ideal}_{\text{TP},\mathcal{S},\Sigma}$ *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* $\mathcal{D}$*:*

$$\Pr[\mathcal{D}(\mathsf{Ideal}_{\mathsf{TP},\mathcal{S},\Sigma}(\lambda)) = 1] - \Pr[\mathcal{D}(\mathsf{Real}_{\mathsf{PIMS},\mathcal{A},\Sigma}(\lambda))] \approx 0$$

Theorem 1. PIMS *satisfies with Definition 11 given the existence of CP-SNARK, Pedersen commitment scheme, the digital signature scheme, and the verifiable data registry.*

5 Construction

5.1 Main idea

In this section, we describe a generalized PIMS construction. In other verifiable credential data models [12], the user generates the verifiable presentation via composing the given verifiable credentials from the issuer, whereas, our construction allows that Holder has the flexibility to create verifiable presentations other than the way of assembling issued verifiable credentials. Through utilizing zk-SNARKs as building blocks of the construction, Holder enables applying complex computation to generating the verifiable presentation other than proving knowledge of the verifiable credential. Our protocol performs zkCL, zkVC and zkVP phases as follows. Firstly, Holder commits a claim of Subject then sends the commitment of the claim and its claim value to Issuer. Note that a secret value(opening value of the commitment) of Subject is kept only in Holder to prevent Issuer generating a verifiable presentation without Subject's permission. When Issuer is given the claim value and its commitment, Issuer validates the authenticity of the claim and signs the commitment if the claim is valid. The commitment and its signature of Issuer get to be a verifiable credential. The verifiable credential are included in DID documents in the verifiable data registry. After the verifiable credential is uploaded, Holder generates a zero-knowledge verifiable presentation using the Commit and Prove zk-SNARKs(CP-SNARKs). Specifically, Holder runs a given function from Verifier and computes an output from the verified claim data. For instance, if the Verifier requests Holder to prove that an income level of Subject is in the specific range, Holder runs the function for income range taking as input the claim data and outputs whether the income level is in the range or not. The function output is used for the verifiable presentation value and the CP-SNARKs that proves the function execution's validity is used for its proof.

The CP-SNARKs proves that a commitment is constructed well and the user-generated credential data originate from the knowledge of the commitment without revealing the knowledge. Naively, the CP-SNARKs can be constructed simply if all commitment checks and the presentation generation processes are included in the proof relation of CP-SNARKs. However, if hash algorithms(e.g. SHA-256, MD5) that are composed of heavy computation are included in the proof relation of CP-SNARKs, it can cause inefficiency in proof time, setup time, and crs size. Thus, we optimize the proof relation via the way the commitment's correctness is checked outside the proof relation, and only the equivalence of the value implied in commitment is verified in the proof relation [11]. Specifically, if

the commitment is constructed using the exponentiation(e.g. Pedersen commitment), the commitment can be used as an I/O of the proof relation efficiently. The CP-SNARKs can be constructed using the combination of Π algorithms and $\mathsf{CP}_{\mathsf{link}}$ algorithm that proves commitment validity outside the proof relation. Via the $\mathsf{CP}_{\mathsf{link}}$ algorithm, we remove the commitment check relation from the commit-and-prove relation R_{CP}. The proof relation is composed of the credential function and a commitment equivalency check that identifies whether the commitment used in the credential function is the same as the one used in $\mathsf{CP}_{\mathsf{link}}$ or not. We describe $\mathsf{CP}.\mathsf{link}$ construction in Algorithm 5 and zkCL, zkVC and zkVP constructions in Algorithm 6, 7 and 8 respectively.

5.2 $\mathsf{CP}_{\mathsf{link}}$ construction

We adopt $\mathsf{CP}_{\mathsf{link}}$ construction of Dario.et.al [14]. Intuitively, a commitment has a form of exponentiation of the group element and the proof has the same exponentiation as the commitment. Verifier checks if the proof has the same exponentiation as the exponentiation of the commitment using a bilinear pairing function. $\mathsf{CP}_{\mathsf{link}}$ is composed of tuple of algorithms $\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$.

Setup takes a public parameter pp and a common reference string crs_k^*. crs_k^* is a common reference string for relation R_k^* as input. We only use elements $\mathbf{F}$ that is for I/O components from crs_k^*(N is the maximum number of attributes in one claim value, M is the maximum number of credentials). The elements is used as a linkage of the proof of $\mathsf{CP}_{\mathsf{link}}$ and the proof of the credential generation. The setup algorithm picks random values u,v,w from $\mathbb{Z}_p$ and sets $U = g_2^u$,$V = g_2^v$, and $W = g_2^w$. Then the algorithm samples $M * N$ random values $r_{j,i}$ and computes $t_{j,i} = g_{j,i}^u r_{j,i}^v f_{j,i}^w$ where $g_{j,i}$ is an element of commitment key ck and $j \in [1, M], i \in [0, N]$. Setup sets crs_{link_k} to $(ck, \mathbf{R}, \mathbf{T}, U, V, W)$ and returns crs_{link_k}.

In Prove, the commitment crs_{link_k}, c_x that is a set of knowledge of the commitment on the group elements $f_{j,i}$, the knowledge matrix $\mathbf{u}$, the commitment opening set $\vec{o} = (o_1, ..., o_l)$ are given as input where l is the number of credentials that will be used in the presentation and the size of $\mathbf{u}$ is $l * N$. The algorithm computes proofs T_x and R_x that imply the knowledge of the commitments. The proofs are the result of the exponentiation of knowledge on a different base. Prove sets T_x,R_x to π_{link} and returns the proof.

Verify takes as input a common reference string crs_{link_k}, a linkage value c_x, a commitment set $\vec{c}$, and the proof π_{link}. The algorithm checks the equivalency of the exponent of commitments and the exponent of the proofs using the bilinear pairing function.

5.3 Pedersen vector commitment based PIMS construction

Algorithm 6,7 and 8 describe our zkCL,zkVC and zkVP constructions respectively. Pedersen vector commitment scheme is used for the verifiable credential and CP-SNARKs is used for the verifiable presentation generation. In this construction, CP-SNARKs is split into $\mathsf{CP}_{\mathsf{link}}$ and a zk-SNARK scheme Π. A relation R_k^* checks only the validity of the presentation function execution result and does

Algorithm 5 $\mathsf{CP}_{\mathsf{link}}$ construction

$\mathsf{Setup}(pp, crs_k^*)$

$ck = (\mathbf{G} = \begin{bmatrix} g_{1,0} & \cdots & g_{1,N} \\ \vdots & \ddots & \vdots \\ g_{M,0} & \cdots & g_{M,N} \end{bmatrix}_{g_{m,n} \in \mathbb{G}_1}, g_{2_{\in \mathbb{G}_2}})$

$u, v, w \xleftarrow{\$} \mathbb{Z}_{p^*}$

$U = {g_2}^u, V = {g_2}^v, W = {g_2}^w$

parse $\mathbf{F} = \begin{bmatrix} f_{1,0} & \cdots & f_{1,N} \\ \vdots & \ddots & \vdots \\ f_{M,0} & \cdots & f_{M,N} \end{bmatrix}_{f_{m,n} \in \mathbb{G}_1}$ from crs_k^*

for $j < M + 1$ **do**
  for $i < N + 1$ **do**
    $r_{j,i} \xleftarrow{\$} \mathbb{G}_1$
    $t_{j,i} = g_{j,i}^u r_{j,i}^v f_{j,i}^w$
  end for
end for

$\mathbf{R} = \begin{bmatrix} r_{1,0} & \cdots & r_{1,N} \\ \vdots & \ddots & \vdots \\ r_{M,0} & \cdots & r_{M,N} \end{bmatrix}$

$\mathbf{T} = \begin{bmatrix} t_{1,0} & \cdots & t_{1,N} \\ \vdots & \ddots & \vdots \\ t_{M,0} & \cdots & t_{M,N} \end{bmatrix}$

$crs_{\mathsf{link}_k} = (ck, \mathbf{R}, \mathbf{T}, U, V, W)$

return crs_{link_i}

$\mathsf{Prove}(crs_{\mathsf{link}_i}, c_x, \mathbf{u}, \vec{o})$

parse $\mathbf{u} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,N} \\ \vdots & \ddots & \vdots \\ u_{l,1} & \cdots & u_{l,N} \end{bmatrix}_{l \in [1,M]}$, $\vec{o} = (o_1, ...o_l)_{l \in [1,M]}$

for $j < l + 1$ **do**
  $t_{x_j} = t_{j,0}^{o_j} \prod_{i=1}^{N} t_{j,i}^{u_{j,i}}$
  $r_{x_j} = r_{j,0}^{o_j} \prod_{i=1}^{N} r_{j,i}^{u_{j,i}}$
end for

$T_x = \prod_{j=1}^{l} t_{x_j}$

$R_x = \prod_{j=1}^{l} r_{x_j}$

$\pi_{\mathsf{link}} = (T_x, R_x)$

return π_{link}

$\mathsf{Verify}(crs_{\mathsf{link}_i}, \vec{c}, c_x, \pi_{\mathsf{link}})$

$\pi_{\mathsf{link}} = (T_x, R_x)$

if $e(T_x, g_2) = e(\prod_{j=1}^{l} c_j, U) e(R_x, V) e(c_x, W)$ **then return** 1

else return 0

end if

Algorithm 6 zkCL construction

Setup(λ, N, M)

$pp = \{\mathbf{G} = \begin{bmatrix} g_{1,0} & \cdots & g_{1,N} \\ \vdots & \ddots & \vdots \\ g_{M,0} & \cdots & g_{M,N} \end{bmatrix}, g_2, h_1, ..., h_M\}_{g_{j,i} \in \mathbb{G}_1, (g_2, h_j) \in \mathbb{G}_2}$

return pp

Gen$(pp, \vec{u_j})$

$o_j \xleftarrow{\$} \mathbb{Z}_p$

$c_j \leftarrow g_{j,0}^{o_j} \prod_{i=1}^{N} g_{j,i}^{u_{j,i}}$

$\pi_j \leftarrow h_j^{o_j}$

claim $= (\vec{u_j}, c_j, \pi_j)$

return claim

Verify(pp, claim)

parse $\vec{u_j} = (u_{j,1}, ..., u_{j,N})$

authenticate the validity of $\vec{u_j}$

if $e(c_j, h_j) = e(g_{j,0}, \pi_j) e(\prod_{i=1}^{N} g_{j,i}^{u_{j,i}}, h_j)$ **then return 1**

else

return 0

end if

Algorithm 7 zkVC construction

Setup(λ)

$(sk_j, vk_j) \leftarrow$ Sig.Keygen(λ)

return sk_j, vk_j

Gen(pp, claim, sk_j)

if $1 \leftarrow$ zkCL.Verify(pp, claim) **then** $\sigma_j \leftarrow$ Sig.Sign(sk_j, c_j)

else

abort

end if

$\text{cred}_j = (c_j, \sigma_j)$

return cred_j

Verify(vk_j, cred_j)

$b \leftarrow$ Sig.Verify(vk_j, c_j, σ_j)

return b

Algorithm 8 zkVP construction

$\mathsf{Setup}(\lambda, pp, R_k^*)$

$R_k^* = \{(\phi, w) | \phi = (\vec{c}, \mathbf{u}, \vec{o}, in, out), out = f_k(in, \mathbf{u})\}$
$(crs_k^*, \tau_k^*) \leftarrow \Pi.\mathsf{Setup}(R_k^*)$
$crs_{\mathsf{link}_k} \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Setup}(pp, crs_k^*)$
$crs_k = (crs_k^*, crs_{\mathsf{link}_k})$
return crs_k

$\mathsf{Gen}(crs_k, \vec{\mathsf{cred}}, in, \mathbf{u}, \vec{o})$

parse $\vec{\mathsf{cred}} = ((c_1, \cdots, c_l), (\sigma_1, \cdots, \sigma_l))$, $\vec{o} = (o_1, \cdots, o_l)_{l \in [1,M]}$
$\mathbf{u} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,N} \\ \vdots & \ddots & \vdots \\ u_{l,1} & \cdots & u_{l,N} \end{bmatrix}_{l \in [1,M]}$
$out \leftarrow f_k(in, \mathbf{u})$
$\phi = (\vec{c}, \mathbf{u}, \vec{o}, in, out)$
for $j < l + 1$ **do**
$c_{x_j} = f_{j,0}^{o_j} \prod_{i=1}^{N} f_{j,i}^{u_{j,i}}$
end for
$c_x = \prod_{j=1}^{l} c_{x_j}$
$\pi^* \leftarrow \Pi.\mathsf{Prove}(crs_k^*, \phi; w)$
$\pi_{\mathsf{link}} \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Prove}(crs_{\mathsf{link}_k}, c_x, \mathbf{u}, \vec{o})$
$\pi = (\pi^*, \pi_{\mathsf{link}})$
$\mathsf{pres} = (\vec{\mathsf{cred}}, in, out, \pi)$
return pres

$\mathsf{Verify}(crs_k, \mathsf{pres}, \vec{\mathsf{cred}}, \vec{vk})$

parse present $= (\vec{\mathsf{cred}}^*, in, out, \pi)$
if check $\vec{\mathsf{cred}}^* \neq \vec{\mathsf{cred}}$ **then return** 0
end if
for $j < l + 1$ **do**
$b_j \leftarrow \mathsf{zkVC.Verify}(vk_j, \mathsf{cred}_j)$
end for
if all the $b_j \neq 1$ **then abort**
else
$b \leftarrow \Pi.\mathsf{Verify}(crs_k, c_x, in, out, \pi^*) \bigwedge \mathsf{CP}_{\mathsf{link}}.\mathsf{Verify}(crs_{\mathsf{link}_k}, \vec{c}, c_x, \pi_{\mathsf{link}})$
end if
return b

not check the commitment correctness. Specifically, we denote the presentation function as f_k. An algorithm set $\mathsf{zkCL} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify})$ deals with a claim generation and its verification. In $\mathsf{zkCL.Setup}$, $M * N$ group elements of $\mathbb{G}_1$ and $M + 1$ group elements of $\mathbb{G}_2$ are set to public parameters and $g_{j,i_\in \mathbb{G}_1}$ is used for the commitment key. M denotes a maximum number of credentials and N denotes a maximum number of attributes in one claim. Then, Holder generates his/her privacy-preserving claim by committing the attribute value $\vec{u_j}$ on the commitment key. Specifically, Holder runs $\mathsf{zkCL.Gen}$ to generate a claim. $\mathsf{zkCL.Gen}$ takes as input a public parameter pp and a set of attributes $\vec{u_j}$ where j is an index of the issuer. It picks random scalar value $o_j \in \mathbb{Z}_p$ and generates a commitment of the attributes by computing $c_j = g_{j,0}^{o_j} \prod_{i=1}^{N} g_{j,i}^{u_{j,i}} \cdot o_j$ is set to the secret value of Holder and the value is utilized for the commitment opening. Then Holder generates a proof $\pi_j = h^{o_j}$ that proves a correctness of the commitment without providing a commitment opening value o_j. The proof can be verified in a pairing function. $\vec{u_j}, c_j, \pi_j$ is set to the zero-knowledge claim value claim. $\mathsf{zkCL.Verify}$ verifies the claim taking input as pp and claim. The algorithm first authenticates the attribute value set. If all the attribute values are valid, it checks whether $e(g_{j,0}, \pi_j) = e(g_{j,0}, \pi_j) e(\prod_{i=1}^{N} g_{j,i}^{u_{j,i}}, h_j)$ or not. $\mathsf{zkCL.Verify}$ is performed by only Issuer since only the issuer should have permission to identify the attribute data.

An algorithm set $\mathsf{zkVC} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify})$ handles a verifiable credential generation and its verification. In $\mathsf{zkVC.Setup}$, Issuer generates his/her signing key-verification key by running $\mathsf{Sig.Keygen}(\lambda)$. $\mathsf{zkVC.Setup}$ returns the key pair as an output. $\mathsf{zkVC.Gen}$ is also performed by Issuer. The algorithm takes as input a public parameter pp, given claim data $\mathsf{claim} = (\vec{u_j}, c_j, \pi_j)$ and runs $\mathsf{zkCL.Verify}$. If the verification is successful, Issuer generates a signature on c_j. c_j and its signature σ_j are set to a zero-knowledge verifiable credential value cred_j. The algorithm outputs cred_j. cred_j is uploaded to the verifiable data registry so that $\mathsf{Verifier}$ can use the credential data to verify the given presentation value.

An algorithm set $\mathsf{zkVP} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify})$ deals with a verifiable presentation generation and its verification. The verifiable presentation is generated by Holder using CP-SNARK scheme. A common reference string is required to use the CP-SNARKs as building blocks of the verifiable presentation generation. The common reference string should be generated from the trusted authority to use pairing based zk-SNARK [35, 19, 20, 25]. It is possible for the untrusted third party to generate a common reference string if the universal reference string based zk-SNARK [7, 15, 29] is used. However, we assume, for the generality, that the setup process is generated by a trusted party. $\mathsf{zkVP.Setup}$ generates a common reference string for R_k* and a common reference string for $\mathsf{CP}_{\mathsf{link}}$ algorithm respectively. R_k^* describes a relation where the presentation data $out = f_k(in, \mathbf{u})$. Note that the attribute value matrix $\mathbf{u}$ is $l * N$ size of matrix where l is the number of credentials used in the presentation generation. We assume that Holder restructures an attribute matrix $\mathbf{u}$ that will be used in a presentation generation from the whole attribute matrix corresponding to the commitments. Also, for simplicity, we assume that the commitment index j cor-

responds to the index j of the matrix $\mathbf{u}$. The algorithm first generates a crs_k^* by running Π.Setup and then generates crs_{link_k} by running $\mathsf{CP}_{\mathsf{link}}$.Setup that takes as input a commitment key $ck = ((\mathbf{G}, g_2)$ and the common reference string crs_k^*. Though $\mathsf{CP}_{\mathsf{link}}$.Setup needs only the key of I/O in crs_k^*, for simplicity, we remark that $\mathsf{CP}_{\mathsf{link}}$.Setup takes as input crs_k^*. The algorithm runs $\mathsf{CP}_{\mathsf{link}}$.Setup and returns $crs_k = (crs_k^*, crs_{\mathsf{link}})$. zkVP.Gen takes as input a common reference string crs_k, zero-knowledege credential set $\vec{\mathsf{cred}} = (\mathsf{cred}_1, ..., \mathsf{cred}_l)_{l\in[1,M]}$, a presentation function input in and attribute value matrix $\mathbf{u}$, a secret value set of Holder $\vec{o} = (o_1, ..., o_l)_{l\in[1,M]}$. The algorithm computes a presentation data out by running a presentation function f_k. Holder parses $f_{j,i}$ that is the commitment key for I/O value from crs_k^* and generates an intermediate value for $\mathsf{CP}_{\mathsf{link}}$ c_{x_j}. To generate the intermediate value, Holder computes all $c_{x_j} = f_{j,0}^{o_j} \prod_{i=1}^{N} f_{j,i}^{u_{j,i}}$. The intermediate values have the same exponent as the exponent of the commitment c_j. Holder takes π^* by running Π.Prove that proves validity of the presentation generation. Then Holder takes π_{link} by running $\mathsf{CP}_{\mathsf{link}}$.Prove that proves the commitment correctness. The algorithm returns the zero-knowledge verifiable presentation $\mathsf{pres} = (\mathsf{cred}, in, out, \pi = (\pi^*, \pi_{\mathsf{link}}))$. zkVP.Verify takes as input a common reference string crs_k, the zero-knowledge verifiable presentation pres, the zero-knowledge verifiable credential set from the verifiable data registry $\vec{\mathsf{cred}}$ and the signature verification key set $\vec{vk}$. Verifier verifies all the signatures on commitment c_j and verifies π^* and π_{link} only if the all the signatures are verified.

6 Security proof

Algorithm 9 Ideal claim phase idCL

Setup(N, M)
  TP creates an empty PrivateTable
Gen$(\vec{u_j})$
  Holder sends $\vec{u_j} = (u_{j,1}, ..., u_{j,N})$ to TP
  if $\vec{u_j} \in$ PrivateTable **then**
    Obtain $(\vec{u_j}, o_j, c_j)$ from PrivateTable
  else
    TP generates a unique random value c_j, a unique random secret value o_j and inserts $(\vec{u_j}, o_j, c_j)$ in PrivateTable
  end if
  TP sends c_j, o_j to Holder
  Holder outputs claim $= (c_j, \vec{u_j})$
Verify(claim)
  Issuer validates the authenticity of $\vec{u_j}$
  Issuer sends claim to TP
  if $(c_j, \vec{u_j}, o_j) \in$ PrivateTable **then return** 1
  else return 0
  end if

Algorithm 10 Ideal verifiable credential phase idVC

Setup(M)
  TP creates an empty PublicTable
Gen(claim)
  if 1 ← idCL.Verify(claim) **then**
    Issuer sends c_j to TP
    TP inserts (c_j, id_j) to PublicTable
    Issuer sets (c_j, id_j) to cred_j
    TP publishes PublicTable
  else
    abort
  end if
Verify(cred_j)
  if $\mathsf{cred}_j \in$ PublicTable **then return 1**
  else
    return 0
  end if

Proof outline : In this section, We prove Theorem 1. We assume that the simulator $\mathcal{S}$ in ideal execution cannot forge a function output which is notified by the trusted party TP and cannot distinguish the user information $\mathbf{u}$ value when a unique random value c_b is given. Additionally, we show an the adversary $\mathcal{A}$ in real execution can forge the presentation data at most as much the ability as simulator $\mathcal{S}$ and all distributions in real execution are computationally equal to the ideal execution. It means that it is impossible for adversary $\mathcal{A}$ to forge the zero-knowledge verifiable presentation and distinguish the zero-knowledge verifiable credential in real execution if an attack in the ideal execution is impossible. To show the equality of the distributions, we first construct the simulator $\mathcal{S}$ using the adversary $\mathcal{A}$ as a building block and demonstrate that the output of $\mathcal{S}$ is equal to those of $\mathcal{A}$. The simulation for the output of real execution is performed by assuming that distributions of the ideal execution are computationally equal to the distribution of the real execution rather than receiving embedded values from the assumption. In this approach, we define our system in terms of the ideal execution implemented by a trusted party TP, which plays the role of our cryptographic protocol in the real system. In the ideal execution, a collection of parties interacts with the TP according to a specific interface. In the real execution, the parties interact with each other using our protocol.

We first describe the ideal execution idCL, idVC, idVP in Algorithm 9,10 and 11. The ideal execution consists of a set of algorithms $\mathsf{Ideal}_{\mathsf{TP},\mathcal{S},\Sigma} = (\mathsf{idCL}, \mathsf{idVC}, \mathsf{idVP})$. We assume that only the trusted authority TP has permission to write a value on the append-only tables PublicTable, PrivateTable. PublicTable is open to all the entities that join the protocol. On the other hand, PrivateTable is kept only to the TP. idCL deals with the claim generation phase and idVC handles the verifiable credential generation phase and idVP deals with the verifiable presentation generation phase. Each algorithm consists of (Setup, Gen, Verify) as equivalent to

Algorithm 11 Ideal verifiable presentation phase idVP

$\mathsf{Setup}(\cdot)$

TP generates a presentation function f_k

TP notifies f_k to all parties

$\mathsf{Gen}(f_k, \vec{\mathsf{cred}}, in, \mathbf{u}, \vec{o})$

$\vec{\mathsf{cred}} = (\mathsf{cred}_1, \ldots, \mathsf{cred}_l)_{l\in[1,M]}$, $\mathbf{u} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,N} \\ \vdots & \ddots & \vdots \\ u_{l,1} & \cdots & u_{l,N} \end{bmatrix}_{l\in[1,M]}$, $\vec{o} = (o_1, \ldots, o_l)_{l\in[1,M]}$

Holder runs $f_k(in, \mathbf{u})$ and gets a presentation out

Holder sends $(f_k, \mathbf{u}, \vec{o}, in, out)$ to TP

for $j < l + 1$ **do**

  if $(\vec{u_j}, o_j, c_j) \notin$ PrivateTable **then** abort

  end if

end for

TP runs $f_k(in, \mathbf{u})$

if $f_k(in, \vec{u}) \neq out$ **then** abort

end if

Holder sets a presentation $\mathsf{pres} = (\vec{\mathsf{cred}}, f_k, in, out)$

return pres

$\mathsf{Verify}(\mathsf{pres}, \vec{\mathsf{cred}})$

parse $\mathsf{pres} = (\mathsf{cred}^*, f_k, in, out)$

if $\vec{\mathsf{cred}}^* \neq \vec{\mathsf{cred}}$ **then** abort

else

  Verifier sends pres to TP

  TP checks $f_k(in, \mathbf{u}) = out$

  if $f_k(in, \mathbf{u}) = out$ **then** return 1

  else return 0

  end if

end if

those of the real execution. idCL.Setup is operated by TP and TP creates empty private table PrivateTable taking input as the maximum number of attributes in one claim N and the maximum number of credential M. When the trusted party completes the setup, Holder runs idCL.Gen to generate a claim from the attributes. Firstly, Holder sends a set of attributes $\vec{u_j} = (u_{j,1}, \cdots, u_{j,N})$ to the trusted party TP. If the attributes are registered in PrivateTable, then TP obtains $\vec{u_j}$, its opening value o_j and corresponding unique random value c_j from the table. Otherwise, TP generates unique random values c_j and o_j and stores them in PrivateTable with the attribute set $\vec{u_j}$. TP sends the unique random values to Holder and Holder sets $(c_j, \vec{u_j})$ to a claim value. When the claim value claim is sent to the Issuer for validation, Issuer verifies the claim by running idCL.Verify taking input claim. Issuer authenticates the attributes $\vec{u_j}$, and sends the claim to TP. TP checks if the set $(c_j, \vec{u_j}, o_j)$ is in PrivateTable. If the set is in the table, TP returns 1 or 0 otherwise.

idVC describes an ideal execution of the verifiable credential phase. In idVC.Setup, TP creates an empty public table PublicTable. idVC.Gen generates a verifiable credential taking input claim. It is performed by Issuer. The entity verifies the given claim value by running idCL.Verify and sends c_j to TP if idCL.Verify outputs 1. TP then puts c_j with the issuer's id id_j in PublicTable. When the values are inserted in PublicTable, then Issuer sets the values as a verifiable credential cred_j and PublicTable is published. Verifier checks the validity of cred_j by running idVC.Verify. In this algorithm, Verifier checks whether cred_j is in PublicTable or not. If cred_j is in the table, the algorithm returns 1 or 0 otherwise.

idVP handles an ideal execution of the verifiable presentation phase. In idVP.Setup, TP generates a presentation function f_k and notifies f_k to all parties. While the trusted entity generates a public parameter for CP-SNARK in the real execution, the trusted entity generates only the presentation function f_k. idVP.Gen takes as input a function f_k, credential set $\vec{\mathsf{cred}}$, a function input in, an attribute matrix $\mathbf{u}$ and a set of the secret values of Holder $\vec{o}$. As same as the real execution case, we assume for simplicity that the indexes of $\mathbf{u}$ corresponds to those of the commitments. Holder generates a verifiable presentation output out then sends out with the function f_k, the attribute matrix $\mathbf{u}$, the secre value set $\vec{o}$ and the auxiliary input in to TP. TP checks if the attributes and the secret values are recorded in the PrivateTable and checks $f_k(in, \mathbf{u})$ only if the attributes are recorded in advance. If the computation of $f_k(in, \mathbf{u})$ is valid, Holder sets the verifiable presentation $\mathsf{pres} = (\mathsf{cred}, \vec{f_k}, in, out)$. Verifier verifies the presentation by running idVP.Verify. Verifier checks the equality of the given $\vec{\mathsf{cred}}^*$ and $\vec{\mathsf{cred}}$ that is recorded in PublicTable. Verifier queries the correctness of out to TP only if the credential equality test is passed. TP returns 1 if the computation is correct, or 0 otherwise.

We now construct a simulator $\mathcal{S}$ using the adversary of real execution $\mathcal{A}$ as a building block. First, $\mathcal{S}$ runs idCL.Setup and idVP.Setup and acquires pp, crs_k with a trapdoor τ_k^*. It runs an adversary $\mathcal{A}$ and the proof π_{real} are given to $\mathcal{S}$. In the process of running $\mathcal{A}$, the simulator $\mathcal{S}$ can provide simulated proofs π_{sim} by running Π.SimProve that uses the trapdoor. When the proof is given, $\mathcal{S}$ runs

a knowledge extractor $\chi_{\mathcal{A}}$ and extracts $\mathbf{u}, \vec{o}$ from the proof π_{real}. The simulator ensures that all $\vec{u_j}, o_j, c_j$ are in PrivateTable and the corresponding credential set $\vec{\mathsf{cred}}$ is in PublicTable. Then the simulator generates the presentation output *out*. $\mathcal{S}$ finally outputs $\vec{\mathsf{cred}}$,f_k,*in*,*out*. We now show that the distribution of the simulator is computationally indistinguishable from the distribution of real world experiments via a series of hybrid games.

1. G_0 : This is a real execution experiment

2. G_1 : In this game, we replace the proof π_{real} by honest parties with simulated proof π_{sim}. 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 commitments with a unique random value. By Lemma 3 we show that if the commitment scheme is secure, then $G_3 \approx G_2$.

5. G_4 : In this game, we replace all signatures with the identifiers of the issuer given by TP. By Lemma 4 we show that if the signature scheme satisfies unforgeability, then $G_4 \approx G_3$ Note that G_4 represents the ideal world execution. By summation over the previous hybrid games we show that $G_4 \approx G_0$. We conclude our proof sketch by presenting the supporting lemmas.

Lemma 1. *For all PPT adversaries $\mathcal{A}$, if simulation sound CP-SNARK 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 is CP-SNARK that has a efficient simulator. In section 3, we show that the simulator will fail with at most negligible probability. Therefore ϵ is negligible.

Lemma 2. *For all PPT adversaries $\mathcal{A}$, if knowledge extractable CP-SNARK 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 CP-SNARK 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 originally described, but we now replace the commitment with randomly chosen value. By construction Com is a statistically hiding commitment scheme and therefore the probability that an adversary can detect this substitution is negligible.

Lemma 4. *For all PPT adversaries $\mathcal{A}$, the advantage of distinguishing G_3 and G_4 is $\mathcal{A}_{G4} - \mathcal{A}_{G3} \leq \epsilon$ where ϵ is negligible.*

Proof. The simulator continues to operate in the same manner as the real execution, but we now replace the signature with the issuer identifier value given from TP. By construction Sig satisfies the unforgeability and therefore the probability that an adversary can forge the signature is negligible.

6.1 Resistance for replay attacks

For the resistance of the replay attack, the verifier can choose one of the two approaches. One is an approach to enforce the user to create a serial number of verifiable presentations such as Zcash [39]. The serial number can be generated by committing the user secret value and the fresh random value. Specifically, $\mathsf{Com}(o_j||r_j)$ can be a serial number of the verifiable presentation where o_j is the user secret value and r_j is the unique random value. If the proof of the verifiable presentation can guarantee the correctness of the commitment and the serial number is included in the verifiable presentation, the verifier can check the re-use of the verifiable presentation. However, though the cost of the commitment check can be alleviated in the case of using CP-SNARKs, it causes an additional commitment check process anyway. The other approach is that the verifier enforces the user to use simulation-extractable zk-SNARKs to create the verifiable presentation. Since the proof should not be forged even if an adversary can access simulated proofs freely in simulation-extractable zk-SNARKs, the proof has a randomized form. Thus, the verifier can check a one-time-use of the verifiable presentation by requesting the user to register his zk-SNARKs proof to the verifiable data registry or constructing a database that manages the verifiable presentation of the user.

7 Experiment

In this section, we show the performance of PIMS construction by diversifying the proof circuit size and the input size. The proof circuit is composed

Table 2: Experiment environment

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

Table 3: Test information summary

	Case 1	Case 2	Case 3
Number of if-statement	1	100	100
Number of inputs	1 (32 bit int)	100 (32 bit int)	100 (256 bit string)

of "if-statements". Specifically, each proof circuit is composed of a single "if-statement" with equality check and 100 "if-statements" with "greater than or equal to statements". We performed the experiment on an Ubuntu 16.04 LTS 64bit environment with an Intel(R) Core(TM) i5-4670 CPU @ 3.40 GHz Quad Core CPU environment, DDR4 24 GB memory. Note that PIMS consists of zk-SNARK Π for credential proof and the commitment proving algorithm $\mathsf{CP}_{\mathsf{link}}$. We adopt Groth16 construction [19] as zk-SNARK Π. Additionally, we utilize jsnark [26] which uses libsnark [5] as its submodule to generate SNARK circuits.

Table 4: Performance experiment results on different circuit size and inputs

		Case 1	Case 2	Case 3
zkVP.Setup	Π.Setup	0.0129s	0.0501s	5.8479s
	$\mathsf{CP}_{\mathsf{link}}$.Setup	0.0031s	0.0528s	12.8704s
	Total	0.0165s	0.1163s	22.0819s
zkVP.Gen	Π.Prove	0.0038s	0.0181s	1.9121s
	$\mathsf{CP}_{\mathsf{link}}$.Prove	0.0001s	0.0005s	0.0016s
	Total	0.0038s	0.0186s	2.0402s
zkVP.Verify	Π.Verify	0.0011s	0.0011s	0.0014s
	$\mathsf{CP}_{\mathsf{link}}$.Verify	0.0016s	0.0016s	0.0016s
	Total	0.0027s	0.0027s	0.0030s

Table 3 summarizes the test case to diversify the circuit size and the input size. The proof circuit in case 1 is composed of only a single if-statement and a single 32-bit integer input. In case 2 and case 3, both the proof circuits consist of 100 if-statements and 100 inputs, but the input type is different. The input type of the former is an integer, whereas, the input type of the latter is a string.

Table 4 shows the performance results for each case. If the circuit is composed of only a single if-statement, the performance of the credential setup, prove, and verification is quite high. Specifically, the proving time of case 1 takes only 3.8 ms. In case 2, the proof generation time increased by 6 times compared to case 1.

Table 5: Size experiment results on different circuit size and inputs

	Case 1	Case 2	Case 3
crs^*	$6.95KB$	$56.59KB$	$14.35MB$
crs_{link}	$0.89KB$	$14KB$	$3.4MB$
Total	$7.84KB$	$70.59KB$	$17.75MB$
π^*	$0.12KB$	$0.12KB$	$0.12KB$
π_{link}	$64B$	$64B$	$64B$
Total π	$0.18KB$	$0.18KB$	$0.18KB$

Though case 3 has the same number of if-statements and the number of inputs as case 2, the performances are low compared to the case 2 since the input type is a 256-bit string. However, the verification time is almost equivalent in all cases.

Table 5 shows the sizes of the common reference strings and the proofs. The common reference string size increases by the proof circuit size. However, the size of the proof in all cases is equivalent regardless of the proof circuit size. Specifically, the proof size remains as constant ($0.18\ KB$) in all cases.

8 Conclusion

In this paper, we propose a privacy-preserving self-sovereign identity model using zk-SNARKs as building blocks. Our privacy-preserving self-sovereign identity model supports data minimization for any arbitrary relations while remaining the control, persistence of the identity model. Furthermore, the ZKP is optimized in our model utilizing Commit-and-Prove scheme. Via the experiments, we can ascertain the practicality of our proposal. The security of the self-sovereign identity model is formally proven in the random oracle model. In the future, we will extend our self-sovereign identity model to cover the data minimization in terms of Non-Fungible Token and further improve the performance of the zk-SNARKs computation.

References

1. Abraham, A., Theuermann, K., Kirchengast, E.: Qualified eid derivation into a distributed ledger based idm system. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). pp. 1406–1412. IEEE (2018)
2. Allen, C.: The path to self-sovereign identity. Life With Alacrity (2016)
3. Alsayed Kassem, J., Sayeed, S., Marco-Gisbert, H., Pervez, Z., Dahal, K.: Dns-idm: A blockchain identity management system to secure personal data sharing in a network. Applied Sciences **9**(15), 2953 (2019)
4. Baylina, J.: iden3/snarkjs. https://github.com/iden3/snarkjs (2020)
5. Ben-Saason, E., Chiesa, A., Genkin, D., Kfir, S., Tromer, E., Virza, M.: libsnark: C++ library for zksnark proofs (2014)

6. Bowe, S., Gabizon, A.: Making groth's zk-snark simulation extractable in the random oracle model. IACR Cryptol. ePrint Arch. **2018**, 187 (2018)
7. 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)
8. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 677–706. Springer (2020)
9. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: International Conference on Security in Communication Networks. pp. 268–289. Springer (2002)
10. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Annual international cryptology conference. pp. 56–72. Springer (2004)
11. 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)
12. Consortium, W.W.W., et al.: Verifiable credentials data model 1.0: Expressing verifiable information on the web. https://www. w3. org/TR/vc-data-model/?# core-data-model (2019)
13. Ferdous, M.S., Chowdhury, F., Alassafi, M.O.: In search of self-sovereign identity leveraging blockchain technology. IEEE Access **7**, 103059–103079 (2019)
14. Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B.: Hash first, argue later: Adaptive verifiable computations on outsourced data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1304–1316 (2016)
15. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. **2019**, 953 (2019)
16. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: International Conference on Financial Cryptography and Data Security. pp. 81–98. Springer (2016)
17. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 626–645. Springer (2013)
18. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. Journal of the ACM (JACM) **62**(4), 1–64 (2015)
19. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
20. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Annual International Cryptology Conference. pp. 581–612. Springer (2017)
21. Grüner, A., Mühle, A., Meinel, C.: On the relevance of blockchain in identity management. arXiv preprint arXiv:1807.08136 (2018)
22. Hamer, T., Taylor, K., Ng, K.S., Tiu, A.: Private digital identity on blockchain. In: Samavi, R., Consens, M.P., Khatchadourian, S., Nguyen, V., Sheth, A.P., Giménez-García, J.M., Thakkar, H. (eds.) Proceedings of the Blockchain enabled Semantic Web Workshop (BlockSW) and Contextualized Knowledge Graphs (CKG) Workshop co-located with the 18th International Semantic Web Conference,

BlockSW/CKG@ISWC 2019, Auckland, New Zealand, October 27, 2019. CEUR Workshop Proceedings, vol. 2599. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2599/paper5.pdf
23. Johansson, T., Nguyen, P.Q.: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013, Proceedings, vol. 7881. Springer (2013)
24. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: International conference on the theory and application of cryptology and information security. pp. 177–194. Springer (2010)
25. Kim, J., Lee, J., Oh, H.: Simulation-extractable zk-snark with a single verification. IEEE Access **8**, 156569–156581 (2020)
26. Kosba, A., Papamanthou, C., Shi, E.: xjsnark: a framework for efficient verifiable computation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 944–961. IEEE (2018)
27. Lee, J., Kim, J., Oh, H.: Forward-secure multi-user aggregate signatures based on zk-snarks. IEEE Access **9**, 97705–97717 (2021)
28. Lipmaa, H.: Simulation-extractable snarks revisited. Tech. rep., Cryptology ePrint Archive, Report 2019/612, 2019. http://eprint. iacr. org ... (2019)
29. 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)
30. Mühle, A., Grüner, A., Gayvoronskaya, T., Meinel, C.: A survey on essential components of a self-sovereign identity. Computer Science Review **30**, 80–86 (2018)
31. Naik, N., Jenkins, P.: uport open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain. In: 2020 IEEE International Symposium on Systems Engineering (ISSE). pp. 1–7. IEEE (2020)
32. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)
33. Othman, A., Callahan, J.: The horcrux protocol: a method for decentralized biometric-based self-sovereign identity. In: 2018 international joint conference on neural networks (IJCNN). pp. 1–7. IEEE (2018)
34. Otte, P., de Vos, M., Pouwelse, J.: Trustchain: A sybil-resistant scalable blockchain. Future Generation Computer Systems **107**, 770–780 (2020)
35. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)
36. Peng, K., Bao, F.: An efficient range proof scheme. In: 2010 IEEE Second International Conference on Social Computing. pp. 826–833. IEEE (2010)
37. Popov, S.: The tangle. White paper **1**(3) (2018)
38. Reed, D., Sporny, M., Longley, D., Allen, C., Grant, R., Sabadello, M., Holt, J.: Decentralized identifiers (dids) v1. 0: Core architecture, data model, and representations. W3C Working Draft **8** (2020)
39. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE (2014)
40. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)

41. Soltani, R., Nguyen, U.T., An, A.: A new approach to client onboarding using self-sovereign identity and distributed ledger. 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. 1129–1136. IEEE (2018)
42. Stokkink, Q., Epema, D., Pouwelse, J.: A truly self-sovereign identity system. arXiv preprint arXiv:2007.00415 (2020)
43. Stokkink, Q., Pouwelse, J.: 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). pp. 1336–1342. IEEE (2018)
44. Windley, P.: How sovrin works. Sovrin Foundation pp. 1–10 (2016)
45. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)
46. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Annual International Cryptology Conference. pp. 733–764. Springer (2019)
47. Yang, X., Li, W.: A zero-knowledge-proof-based digital identity management scheme in blockchain. Computers & Security **99**, 102050 (2020)
48. Zhu, X., Badr, Y.: Identity management systems for the internet of things: a survey towards blockchain solutions. Sensors **18**(12), 4215 (2018)