Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

# A zero-knowledge-proof-based digital identity management scheme in blockchain

*Xiaohui Yang, Wenjie Li*[*]

*School of Cyber Security and Computer, Hebei University, Baoding 071000, China*

ARTICLE INFO

*Article history:*
Received 30 March 2020
Revised 8 September 2020
Accepted 10 September 2020
Available online 15 September 2020

*Keywords:*
Blockchain
Zero-knowledge proof
Smart contract
Digital identity
Privacy protection

ABSTRACT

The traditional centralized digital identity management system (DIMS) has been subject to threats such as fragmented identity, single point of failure, internal attacks and privacy leakage. Emerging blockchain technology allows DIMSs to be deployed in it, which largely alleviates the problems caused by the centralized third party, but its inherent transparency and lack of privacy pose a huge challenge to DIMSs. In this regard, we leverage the smart contracts and zero-knowledge proof (ZKP) algorithms to improve the existing claim identity model in blockchain to realize the identity unlinkability, effectively avoiding the exposure of the ownership of attributes. Furthermore, we implement a system prototype named BZDIMS that includes a challenge-response protocol, which allows users to selectively disclose their ownership of attributes to service providers to protect users' behavior privacy. Performance evaluation and security analysis show that our scheme achieves effective attribute privacy protection and a wider application scope compared with the prior model.

## 1. Introduction

To meet the different needs of an increasing number of online services, plenty of DIMSs have emerged. However, most DIMSs are centralized, which are fragile and vulnerable to attacks (Dunphy and Petitcolas, 2018), such as single points of failure and phishing attacks (Alpár et al., 2011), malicious tampering (Gaetani et al., 2017) and internal attacks (Zikratov et al., 2017). Also, database information leakage of the centralized DIMS causes great losses to users (Ingram, 2018; Wei, 2018), such as the leak of China Huazhu Hotels Group's 500 million pieces of client information (Wei, 2018), and the privacy data leak scandal of tens of millions of Facebook users (Ingram, 2018). Therefore, centralized DIMS cannot guarantee the confidentiality and availability of identity information (Efanov and Roschin, 2018; Ingram, 2018; Wei, 2018; Ferdous et al., 2019).

Besides, most service providers (SPs) are also identity providers (IdPs). Such a dual character forces users to register under the domain of the SP if they want to access services from the SP. This isolated identity model (Ferdous et al., 2019) has led to a rapid expansion of the number of digital identities on the Internet, generating countless identity fragments that users have been overwhelmed by the innumerable account information. Although federated (Ferdous et al., 2019) and user-centric (Ferdous et al., 2019) identity models have emerged that allow multiple SPs to share the same IdP and attempt to reduce identity fragments, users' authentication operations to access services of other domains are redirected to the IdP, which undoubtedly undermines the behavior privacy.

It is gratifying that the emerging blockchain technology has brought a glimmer of light to solve identity problems (Efanov and Roschin, 2018). The blockchain has a tamper-proof distributed ledger technology, and anyone can host the distributed ledger and record the transaction permanently. Therefore, we can utilize the distributed ledger and the consensus mechanism between nodes to create a trusted distributed DIMS in an untrusted environment to achieve the full

* Corresponding author.
*E-mail addresses:* yxh@hbu.edu.cn (X. Yang), tom643190686@gmail.com (W. Li).

https://doi.org/10.1016/j.cose.2020.102050
0167-4048/© 2020 Elsevier Ltd. All rights reserved.

life cycle of registration, authentication, authorization and revocation, which allows users to manage their identities without relying on the third party, so as to solve the problems of identity fragmentation, theft and single points of failure in traditional centralized DIMSs.

The most widely discussed at present, and the most likely to replace the traditional centralized DIMS is the claim identity model (Al-Bassam, 2017; Zhu and Badr, 2018), which is dedicated to storing the mapping of user identifier to the attribute in the public and transparent distributed ledger. The IdP will verify the attribute value and issue claims on the mapping to endorse the user's ownership of the attribute. There is no doubt that the user can only propagate non-sensitive attribute information to the distributed ledger. Otherwise, it will compromise the user's identity privacy. Privacy identity information cannot be stored in the blockchain, which does limit the application scope of blockchain-based DIMSs, especially when the scenario comes to the commercial or financial industry, customers need to provide enough sensitive information to follow the KYC (Know Your Customer) (Christensen et al., 2016) procedure. For example, when Alice submits a loan application to a bank, and the bank needs to obtain some necessary attribute information of Alice to ensure that Alice has the ability to repay the loan, such as occupation, credit, age, income level and asset information, which undoubtedly contains some sensitive identity information related to personal privacy that should not be exposed. Therefore, how to prevent the ownership of privacy attributes from being public and make the privacy attributes acknowledged by SPs becomes the key to solve this problem.

The main contributions of this paper are as follows:

We improve the existing claim identity model and define the privacy attribute token. The improved claim identity model avoids the exposure of the ownership of the privacy attribute in the public and transparent distributed ledger by including the attribute identifier and user's public key in the hash preimage. And this model realizes the unlinkability between the user and his/her identity, which extends the application scope of the existing claim identity model.

We implement a digital identity management system named BZDIMS to accomplish a complete life cycle of privacy attributes. The efficient and fine-grained management of attributes within BZDIMS provides a good foundation for other security mechanisms. In addition, we share codes of particular computations and smart contracts of BZDIMS on GitHub.[1]

We design a challenge-response protocol that allows the user to selectively disclose the ownership of the attribute to SP. It is worth mentioning that when the user accesses services, the user's authentication operation is redirected to the privacy attributes through zero-knowledge proof instead of being redirected to IdPs. Therefore, the authentication content is only visible to the SP, preserving users' behavior privacy.

The rest of the paper is organized as follows: In Section 2, we describe and mathematically define the existing claim identity model and the improved claim identity model respectively. In Section 3, we introduce the preliminaries involved in BZDIMS. In Section 4, we introduce the proposed system model and each process of managing privacy attributes as well as design principles of two particular computations (claimPK, responseSK). Next, evaluation and analysis of BZDIMS are presented in Section 5. Related works on the blockchain-based DIMS are presented in Section 6. Finally, we summarize BZDIMS and discuss the future work in Section 7.

[1] https://github.com/imamtom/BZDIMS

## 2. The identity model

In this section, we first explain certain terms involved in the identity model. And then we describe and mathematically define the existing claim identity model and the improved claim identity model respectively.

### 2.1. Terminologies

**Entity** The entities are physical or logical objects with distinct existence, which include individuals and collectives (e.g., companies, governments, banks). According to their functionalities, entities are divided into three categories, namely identity providers (IdPs), service providers (SPs) and users.

**Attribute** An attribute is a characteristic of an entity, which consists of a name-value pair. And a qualified attribute for authorization must have two properties: **Possession** and **Endorsement**.

**Identifier** An identifier is an attribute whose value can be used to uniquely identify an entity within a context (Ferdous et al., 2014). In the identity solutions based on blockchain, the blockchain address usually serves as the identifier, which is controlled by the blockchain private key.

**Claim** The claim, usually composed of IdP's digital signature and stored in the smart contract, is a verifiable **endorsement** for the attribute.

**Hashclaim** The hashclaim is a hash value of the user's public key, the attribute identifier and a salt, which can be interpreted as the verifiable attestation that the **possession** of attribute has been transferred to the owner of the public key included in hashclaim's preimage.

**Identity** The identity of an entity is defined as the set of attributes bound to its identifier and corresponding claims or hashclaims. Specially, we assume that the attribute identifier is bound to itself by default but it collects an empty set of claims and hashclaims.

**Profile** When the user wants to access the service from the SP, he/she needs to share the required attributes. For the sake of privacy, the asserted identifier and attributes for authorization is defined as the profile.

### 2.2. The existing claim identity model

As shown in Fig. 1, the user first self-generates multiple attributes that do not compromise privacy and uses the blockchain private key that controls his/her identifier to publish them in blockchain. The smart contract will store the attribute and publisher's address, which could be regarded as the **possession** of the attribute.

After that, the IdPs use the blockchain private key to issue claims on the attributes of qualified users. The claim, as a

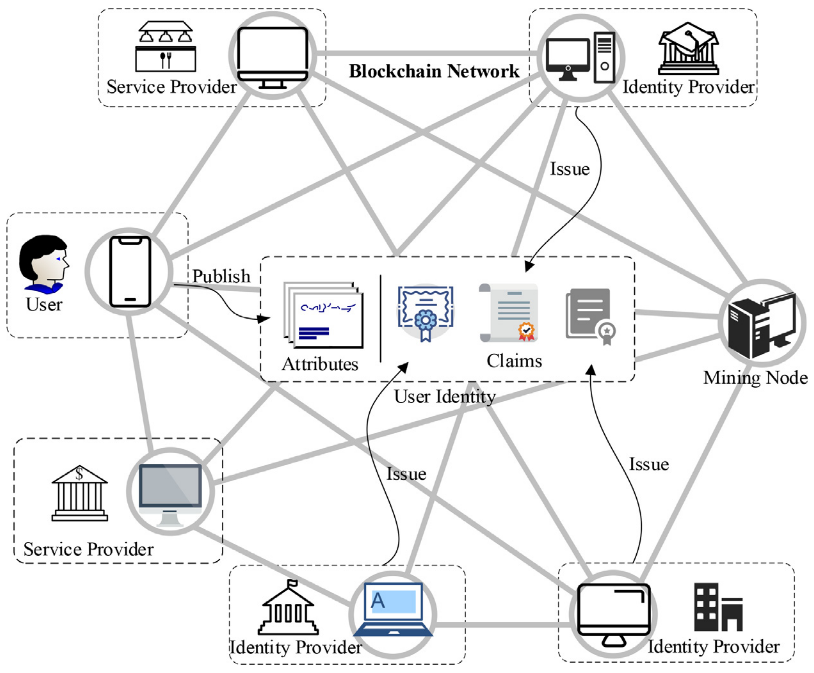


**Fig. 1 – The existing claim identity model.**

verifiable **endorsement**, will be stored in smart contract along with the its issuer's address.

When the user wants to access the service from SP, he/she only needs to demonstrate the ownership of the identifier, usually by making a transfer from the blockchain account wallet. Then the whole identity can be retrieved by SP. Implementation of the existing claim identity model in the Ethereum blockchain can be found in Al-Bassam (2017).

Let us assume that $A$ denotes the set of attributes, $D$ denotes the set of IdPs, $U$ denotes the set of users and $identifier_u$ denotes the identifier of the user $u$. Furthermore, we assume a claim, which is issued on attribute $a$ by the IdP $d$, can be represented by $c_a^d$.

For an attribute $a \in A$, the set of claims issued about $a$, denoted as $AC_a$, is defined as follow:

$$AC_a = \{c_a^{d_1}, c_a^{d_2}, \dots c_a^{d_m}\}(d_1, d_2, \dots d_m \in D)$$

Since not every attribute can receive claims from IdPs, $AC_a$ could be an empty set.

For a user $u \in U$, the set of attributes possessed by $u$ is defined as $A_u$. And the identity of $u$, denoted as $identity_u$, is defined as follow:

$$identity_u = < identifier_u, \{(a, AC_a) | a \in A_u\} >$$

### 2.3. *The improved claim identity model*

As shown in Fig. 2, in the improved claim model, the privacy attribute containing a name-value pair is created by IdP, then the smart contract will store the mapping of the attribute to IdP's identifier. Hence, the IdP's operation of publishing the privacy attribute could be regarded as the **endorsement** of the privacy attribute.

After that, the IdP issues a hashclaim about this privacy attribute by ZKP. The hashclaim attests that **possession** of this attribute has been transferred to the user, which will be stored in the smart contract. The details will be explained in Section 4. The process of issuing the hashclaim will not disclose the user's public key. From the perspective of other entities in blockchain, IdP just issues a hashclaim on a self-created attribute, but other entities did not know which user the IdP transferred the **possession** of this attribute to.

When the user wants to access the service from SP, he/she could assert the ownership of the privacy attribute. Since the **endorsement** of privacy attribute could be checked in smart contracts, the user next needs to attest to the **possession** of attribute and identifier via ZKP.

To distinguish privacy attributes from non-privacy attributes of the prior model, we assume that $M$ denotes the set of privacy attributes. Furthermore, a hashclaim, which is issued on the privacy attribute $m$ by IdP $d$, can be represented by $h_m^d$, whose preimage contains the user's public key.

For a user $u \in U$, the set of privacy attributes possessed by $u$ is defined as $M_u$. Thus the identity of $u$ is defined as follow:

$$identity_u = < identifier_u, \{(m, h_m^d) | m \in M_u, d \in D\} >$$

### 2.4. *Comparison*

The relation between user, identifier and identity is shown in Fig. 3. And the solid arrows indicate that the relation is globally visible, and the dotted arrows indicate that the relation is not visible to other entities.

It could be found that each entity uses one blockchain address that is controlled by a private key to uniquely identifies itself in both two blockchain-based identity models.

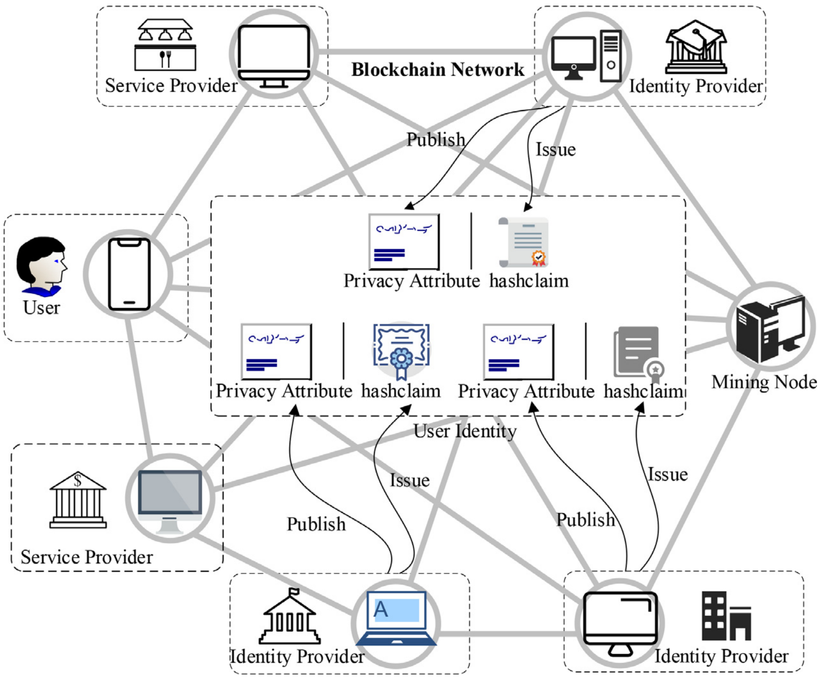
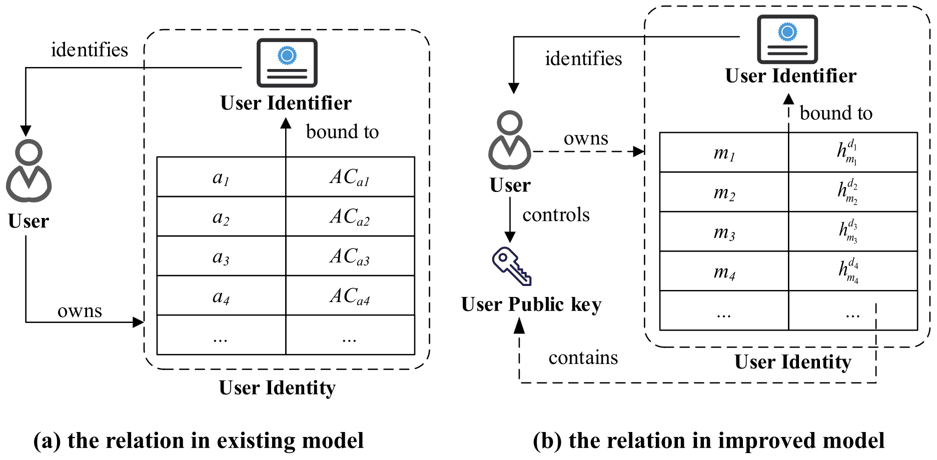


Fig. 2 – The improved claim identity model.



| $a_1$ | $AC_{a1}$ |
|---|---|
| $a_2$ | $AC_{a2}$ |
| $a_3$ | $AC_{a3}$ |
| $a_4$ | $AC_{a4}$ |
| ... | ... |

| $m_1$ | $h_{m_1}^{d_1}$ |
|---|---|
| $m_2$ | $h_{m_2}^{d_2}$ |
| $m_3$ | $h_{m_3}^{d_3}$ |
| $m_4$ | $h_{m_4}^{d_4}$ |
| ... | ... |

Fig. 3 – The relation between user, identifier and identity.

However, in the existing claim identity model, the user's action to publish attributes in blockchain has exposed his/her **possession** of these attributes. The **endorsement** for attribute comes from the claim issued by IdPs. Therefore, the ownership between user and identity in existing claim identity model is visible to other entities as shown in Fig. 3(a).

By contrast, in the improved claim identity model, the **endorsement** for the privacy attribute comes from IdPs' action to create it. By leveraging ZKP, user's **possession** of the attribute is hidden in the verifiable hashclaim, whose preimage includes the user public key that is invisible. Thus what attributes the user has is also unknown as shown in Fig. 3(b), realizing effective privacy protection.

In addition, when accessing the service from SPs, the user in the existing claim model only needs to assert and attest to the fact that he/she possesses a certain identifier. But the user in the improved model needs to follow the challenge-response protocol to assert that he/she possesses a certain identifier and a set of required attributes and demonstrate that preimages of the corresponding hashclaims include his/her public key by ZK-SANRK.

## 3. Preliminaries

In this section, we review some technical preliminaries required in this paper.

## 3.1. *Blockchain*

Since the birth of Bitcoin (Nakamoto, 2019) in 2009, cryptocurrencies have brought a huge impact on traditional finance. Blockchain, as the core technology of Bitcoin, is a distributed ledger technology. Peers in blockchain connect the blocks in chronological sequence into a specific data structure in a chain manner and use cryptography to guarantee strong unforgeability and integrity. Since everyone can host the distributed ledger, the transaction does not depend on intermediaries. The transaction process is transparent, traceable and tamper-proof, thereby establishing a robust trust system in a distributed and trustless scenario.

Besides, according to whether peers need authorization when joining or exiting the blockchain system, blockchains can be divided into two types: permissionless blockchains and permissioned blockchains.

Permissionless blockchain, namely public blockchain, is a completely decentralized blockchain. Public blockchain verifies transactions and incentives in mutually unknown networks based on consensus, thus establishing a completely decentralized trust mechanism.

The permissioned blockchain is divided into a consortium blockchain and a private blockchain. Generally, the consortium blockchain is jointly managed by several institutions and can determine the degree of openness to the public according to the application scenario. Since fewer nodes are participating in the consensus, the consortium blockchain does not use the Proof of Work (PoW) consensus, but mostly uses consensuses such as PoS (Mingxiao et al., 2017) (Proof of Stake), PBFT (Castro and Liskov, 1999) (Practical Byzantine Fault Tolerant) or RAFT (Mingxiao et al., 2017) Algorithm. The transaction per second (TPS) and confirmation time of the consortium chain are significantly different from the public blockchain. Private chains generally belong to individuals or single institutions. Because developers do not want others to join, this blockchain maintained by a single node is a centralized data structure and has a narrow application range.

## 3.2. *Smart contract*

Smart contracts, deployed in blockchain, are automatically executing codes, which have flexible and programmable features and do not rely on any intermediaries. These smart contracts are created and called by the way that entities send transactions, thus preventing unilateral tampering with the rules. With the help of smart contracts, the entity's operation process is made public, and the underlying consensus mechanism of the blockchain guarantees the correctness and consistency of the transaction results. Therefore, smart contracts can be used to define more complex transactions, allowing us to deploy and run DApps (Decentralized Applications) in blockchain, such as crowdfunding, education, and healthcare, rather than just creating more types of cryptocurrencies. So the emergence of smart contracts is also known as the era of blockchain 2.0.

At present, two major blockchain platforms support smart contracts: Ethereum (Wood, 2014) and IBM's HyperLedger (Androulaki et al., 2018). Among them, Ethereum has designed a Turing-complete Ethereum Virtual Machine (EVM) to implement the functions of smart contracts and allows developers to develop smart contracts by using the high-level language Solidity.

## 3.3. *Zero-knowledge proof*

Zero-knowledge proof (ZKP) is a cryptographic technique, which means that the prover can convince the verifier that a certain statement is correct without providing the verifier with any additional information or leaking any information about the witness. The statement could be that the prover knows the preimage of the hash value, or that the prover knows a member of a Merkle tree with a known Merkle root. The ZKP algorithms include zk-SNARKs (Zero-Knowledge Succinct Non-interactive ARguments of Knowledge) (Ben-Sasson et al., 2014), ZKBoo (Giacomelli et al., 2016), zk-STARKs (Ben-Sasson et al., 2018), etc. And lots of facts prove that using ZKP for verification will solve many problems effectively (Morais et al., 2019).

According to whether there is a challenge-response interaction, ZKP algorithms can be divided into two kinds: interactive and non-interactive (Lesavre et al., 2019). In this paper, we would like to employ zk-SNARK to achieve privacy in blockchain because of its minimal calculation for verification and succinct proof. With zk-SNARK, the prover can convince the verifier that they have correctly performed a calculation on a set of inputs without revealing some of the inputs. And the information required for verification is much smaller than calculation.

In all, zk-SNARK enables us to achieve the design of privately transferring the **possession** of the attribute in public blockchain with minimal computational overhead.

## 3.4. *ZoKrates*

ZoKrates (Eberhardt and Tai, 2018), based on zk-SNARK algorithm Groth17 (Groth and Maller, 2017), provides a processing model for off-chain computing and on-chain verifying in Ethereum. More importantly, ZoKrates is a toolbox that consists of a domain-specific language (DSL), a compiler and generators for witness and proof. It is worth mentioning that a Zokrates-based project developed by Ernst & Young is named nightfall (Konda et al., 2019). This project can be directly deployed and run on Ethereum, enabling private transfers without changing the Ethereum stack.

Thus we leverage zk-SNARK in a black-box manner within Zokrates to achieve our work. To be specific, we can complete the operations shown in Fig. 4 with the help of ZoKrates. And the details of operations are as follow:

We could develop particular computations in a human-readable way, then get the DSL file of high-level codes.

Then we compile the DSL file into flattened code which is an abstraction of constraints i.e. circuit. The DSL file could be converted into Rank-1-Constraint-Systems (R1CS) and hence has compatibility with zk-SNARK proof systems.

Same as other zk-SNARK algorithms, ZoKrates performs the setup phase to share Common Reference String (CRS), which results in two public keys, a long proving key and a short verification key. The verification key will be deployed

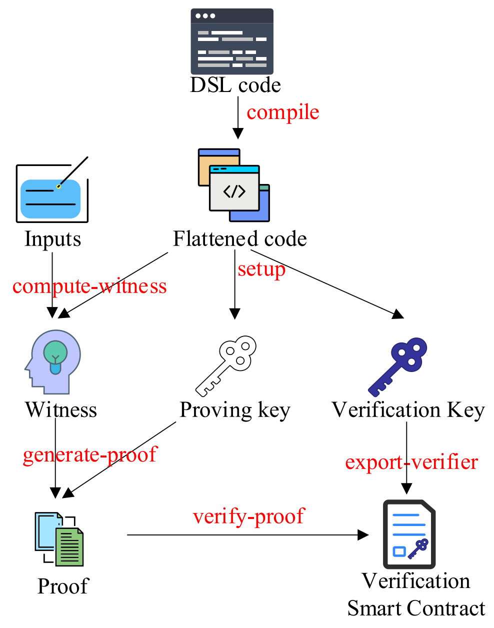


**Fig. 4 – The process of Zokrates.**

in blockchain together with the verification contract and the proving key will be distributed to each prover.

Before generating the zk-SNARK proof, the prover needs to feed public inputs and private inputs into the generator to compute the zk-SNARK witness which satisfies the flattened code.

Then, zk-SNARK proof can be computed against the witness and CRS (i.e. proving key). The proof combined with the public inputs could be verified against the verification key which has been stored in the verification contract. The returned verification result will participate in identity management in smart contracts.

## 4. Architecture and main procedures

In this section, we first present an overview of the proposed scheme and then describe the construction and principle of four procedures of the life cycle of the privacy attribute in detail. Finally, we explain how the challenge-response protocol works.

### 4.1. *Scheme overview*

To implement the improved claim identity model, we propose a Blockchain-and-ZKP-based digital identity management system (BZDIMS) with the help of zk-SNARK and Ethereum. And the proposed system model is presented in Fig. 5.

There are three entities in our scheme, namely Identity provider, Service provider and User.

Identity provider(IdP): Responsible for publishing and revoking the privacy attributes, issuing verifiable hashclaims on them.

Service providers(SP): Responsible for providing online or offline services whose access policy contains requirements for attributes and validating the profile provided by the user.

User: Possessing identity attributes stored in blockchain, accessing the service from IdPs by proving ownership of his/her identifier and attributes.

The privacy attributes are abstracted as tokens in this paper. Besides, five smart contracts are developed to achieve the full life cycle of the privacy attributes including Entity Register Contract (ERC), Attribute Repository Contract (ARC), Knowledge Management Contract (KMC), claimPK Verify Contract (CVC) and responseSK Verify Contract (RVC).

ERC provides an interface *entityRegister*() for entities to register public keys and an interface *entityQuery*() for querying entity public key information. In addition, four functions designed to be called by entities to manage attributes are ARC.*tokenCreate*(), ARC.*tokenRevoke*(), KMC.*tokenClaim*(), and KMC.*tokenResponse*(). The function call relationship between contracts is shown in Fig. 5, where KMC will call the functions in ARC, RVC, and CVC to manage attributes.

In addition, since ZoKrates cannot directly derive the public key from the Ethereum private key based on secp256k1, we use sha256 in the standard library of ZoKrates instead of secp256k1. We define the key pair generated by sha256 as a ZK key pair, where the ZK public key will be included in the hashclaim's preimage, and the ZK private key needs to be properly kept.

Before the solution runs, BZDIMS needs to deploy smart contracts and execute the zk-SNARK setup phase to share the proving key with all. The entities need to register their ZK public key withblockchain address in ERC.

As shown in Fig. 6, the challenge-response protocol consists of five steps, and the full life cycle of privacy attributes includes the following four procedures:

**Creation**: Creation is the initial step in which a privacy attribute token that contains a name-value pair is published by the IdP and the ERC will record the creator's address as the endorsement of the attribute.

**Transferring**: To secretly transfer the possession of the privacy attribute to user in blockchain, IdP needs to issue the hashclaim on this token and prove that it performed the particular computation named claimPK that hashclaim is hashed from token identifier and user ZK public key by ZKP.

**Responding**: To demonstrate the possession of the privacy attribute token, the user needs to prove that he/she owns the ZK private key corresponding to the ZK public key contained in the preimage of hashclaim. The particular computation performed by the user during this process is named responseSK. This step is also a part of the challenge-response protocol.

**Revocation**: The final step is the revocation process which allows the IdP to block the responding of the attribute by modifying the existence field of the token.

Other subsections (4.2, 4.3, 4.4, 4.5, 4.6) will discuss the details about the challenge-response protocol and four procedures of BZDIMS. Used notations and functions are listed respectively in Tables 1 and 2.

### 4.2. *Phase 1: creation*

To create the privacy attribute token, the IdP needs to follow the creation procedure as presented in Fig. 7.

First, the IdP sends a transaction to call *tokenCreate*() in ARC according to the metadata format shown in Table 3. The

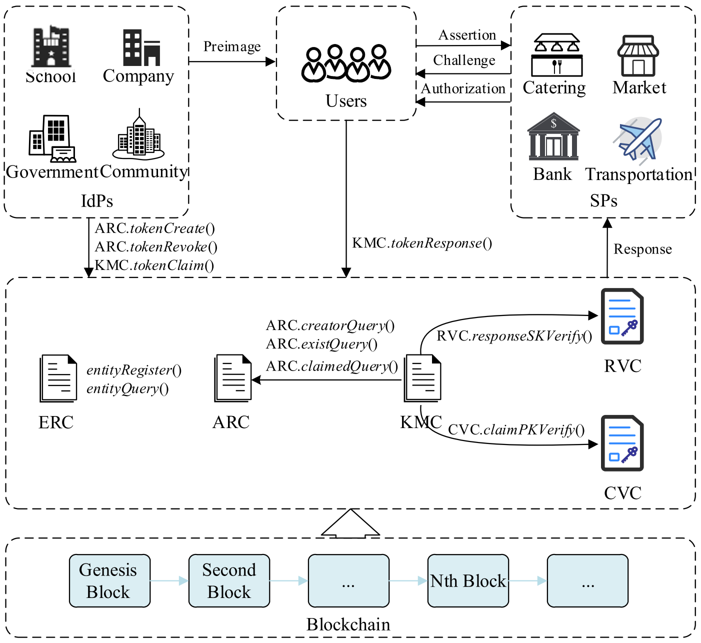


**Fig. 5 – System model.**

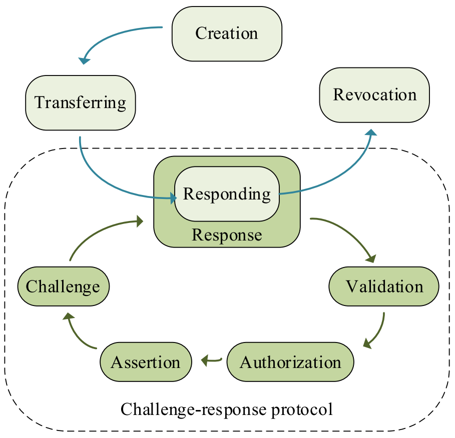


**Fig. 6 – Four procedures of BZDIMS and the challenge-response protocol.**

**Table 1 – Summary of notations.**

| Notation | Meanings | Notation | Meanings |
|---|---|---|---|
| $E$ | Blockchain address | $\Xi$ | One-off blockchain address |
| $pk^E$ | Public key of blockchain address | $sk^E$ | Private key of blockchain address |
| $pk^\Xi$ | Public key of One-off Blockchain address | $sk^\Xi$ | Private key of One-off Blockchain address |
| $pk^Z$ | ZK public key | $sk^Z$ | ZK private key |
| $p$ | Proving key | $vk$ | Verification key |
| $\sigma$ | zk-SNARK Proof | $\psi$ | zk-SNARK Witness |

**Table 2 – Functions in challenge-response protocol.**

| Function | Definition |
|---|---|
| *contentEnc(plaintext, pk)* | Using *pk* to encrypt *plaintext* |
| *contentDec(ciphertext, sk)* | Using *sk* to decrypt *ciphertext* |
| *signatureGen(message, sk)* | Using *sk* to sign *message* |
| *signatureVerify(message, signature, pk)* | Using *pk* to verify *signature* with *message* |

transaction is signed by the blockchain private key $sk^E_{idp}$ that controls the blockchain address $E_{idp}$, so that the sender of the transaction could be recorded as the publisher of the privacy attribute token. As described in Algorithm 1, the *tokenCreate*() will compute a unique ‘tokenId’ for each privacy attribute token for retrieval and store the mapping from tokenId to attribute.

Given the cost of storing data in blockchain is very expensive, and storing large data will affect the system throughput and block synchronization speed (Schäffer et al., 2019), the scheme allows off-chain storage (such as IPFS) for large data (such as the PGP key) to optimize the system performance while providing the ‘hash’ to ensure the attribute authenticity.

The field of ‘name’ is used to record the attribute type, such as role, certificate, qualification, or marked as IPFS. The field of ‘value’ is responsible for recording the content of the attribute or the IPFS address. The field of ‘creator’ is responsible for recording the publisher of the attribute, which could be regarded as a verifiable endorsement from the IdP.

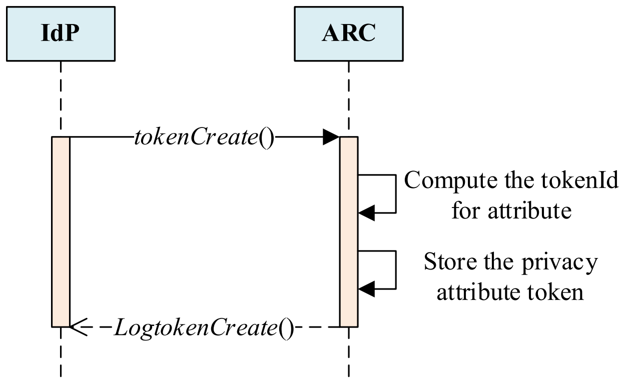

IdP
ARC
tokenCreate()
Compute the tokenId for attribute
Store the privacy attribute token
LogtokenCreate()

**Fig. 7 – Sequence diagram of creation.**

**Table 3 – Privacy attribute token metadata format.**

| Field | Meaning |
|---|---|
| name | The attribute name/IPFS |
| value | The attribute value/address of IPFS |
| hash | The hash of 'name' and 'value' |
| creator | The creator of the token |
| exist | The existence of the token |
| claimed | Whether the token is issued the hashclaim |

**Algorithm 1 – tokenCreate()**

```
Inputs: $\_name, \_value, \_hash$
Outputs: LogTokenCreate()
  tokenId = sha256(msg.sender || timestamp || _name || _value || _hash)
  //create the mapping from tokenId to attribute
  attrTokens[tokenId].name, value, hash = _name, _value, _hash
  attrTokens[tokenId].creator = msg.sender
  attrTokens[tokenId].exist = True
  attrTokens[tokenId].claimed = False
  Emit the event LogTokenCreate(tokenId, _name, _value, msg.sender, _hash)
```

The field of 'exist' is *True* once the token is created. When the token is revoked, it will be set to *False* to block the token's response. The field of 'claimed' is *False* when the token is created. Only after the IdP completes the subsequent process of issuing the hashclaim on this token, the field of 'claimed' will be set to *True*. And this token cannot be issued the hashclaim again if 'claimed' is *True*, preventing the IdP from transferring the attribute possession to multiple users.

Last, the creation of the privacy attribute token will trigger the event *LogTokenCreate*() to notify the IdP completion status. At this time, the IdP has completed the creation, but the attribute possession has not been transferred to the qualified user.

## 4.3. *Phase 2: transferring*

### 4.3.1. *Particular computation of claimPK*

To transfer the possession of attribute created in the phase of creation to the user $u$, the IdP needs to include the identifier of the privacy attribute token and the ZK public key of $u$ into the preimage of the hashclaim. To prevent the attacker from obtaining the ownership between $u$ and the attribute by enumerating ZK public keys registered in ERC and token identifiers to crack the hashclaim, an additional token salt figure is also contained in the preimage.

We assume that the identifier of the privacy attribute token is $\tau$, the ZK public key queried from ERC is $pk_u^Z$, the token salt figure is $\varepsilon_\tau$, and the hashclaim as the sha256 result of these three values is $H_\tau$. The claimPK could be abstracted to the following formula:

$H_\tau$ equals $sha256(\tau||pk_u^Z||\varepsilon_\tau)$

The $pk_u^Z$ and $\varepsilon_\tau$ are private inputs, $\tau$ and $H_\tau$ are public inputs of claimPK. The Zokrates will perform the following calculations:

a) compute $H_\tau{}' := sha256(\tau||pk_u^Z||\varepsilon_\tau)$
b) check $H_\tau{}' == H_\tau$
c) return True

The validity of $\tau$ is verified by the contract. Besides, to verify the correctness of the private inputs, the private inputs or the values computed from the private inputs needs to be checked with at least one public input during the check phase. So when the private input provides the correct ZK public key and the token salt figure, the calculated $H_\tau$' is equal to the public input $H_\tau$.

### 4.3.2. *The procedure of transferring*

The set of constraints of claimPK $Cst_{\text{claim}}$ has already been compiled locally. The generators for zk-SANRK witnesses and proofs are abstracted to *witnessGen*() and *proofGen*() respectively within Zokrates. The procedure of issuing the hashclaim can be summarized as presented in Fig. 8.

First, the IdP computes hashclaim $H_\tau := sha256(\tau||pk_u^Z||\varepsilon_\tau)$ and feeds public inputs $pubInp_{\text{claim}} = (\tau, H_\tau)$ and private inputs $pubInp_{\text{claim}} = (pk_u^Z, \varepsilon_\tau)$ for the circuit of claimPK to generate witness$\psi_{\text{claim}} = witnessGen(Cst_{\text{claim}}, pubInp_{\text{claim}}, priInp_{\text{claim}})$. According to claimPK proving key $p_{\text{claim}}$, the IdP generates the zk-SNARK proof $\sigma_{\text{claim}} = proofGen(p_{\text{claim}}, \psi_{\text{claim}})$. The IdP then calls the *tokenClaim*() in KMC by sending a transaction signed by $sk_{idp}^E$ from address $E_{idp}$ to verify the $pubInp_{\text{claim}}$ and $\sigma_{\text{claim}}$. The details of *tokenClaim*() are shown in Algorithm 2.

Then the IdP watches the triggered event *LogTokenClaim*(). If the event is successfully emitted, the IdP will store the related information $\{\tau, E_u, pk_u^Z, \varepsilon_\tau\}$ in the local database. Finally, the IdP sends the preimage of $H_\tau$ to $u$ through the secure channel. The user $u$ follows the following steps to confirm preimage information and stores $\{\tau, H_\tau \varepsilon_\tau\}$ in the local database.

a) Compute $sha256(\tau||pk_u^Z||\varepsilon_\tau) \rightarrow H_\tau$
b) Check $H_\tau$ is already in the list claimList
c) Check the 'claimed' of $\tau$ is *True*
d) Check the 'exist' of $\tau$ is *True*

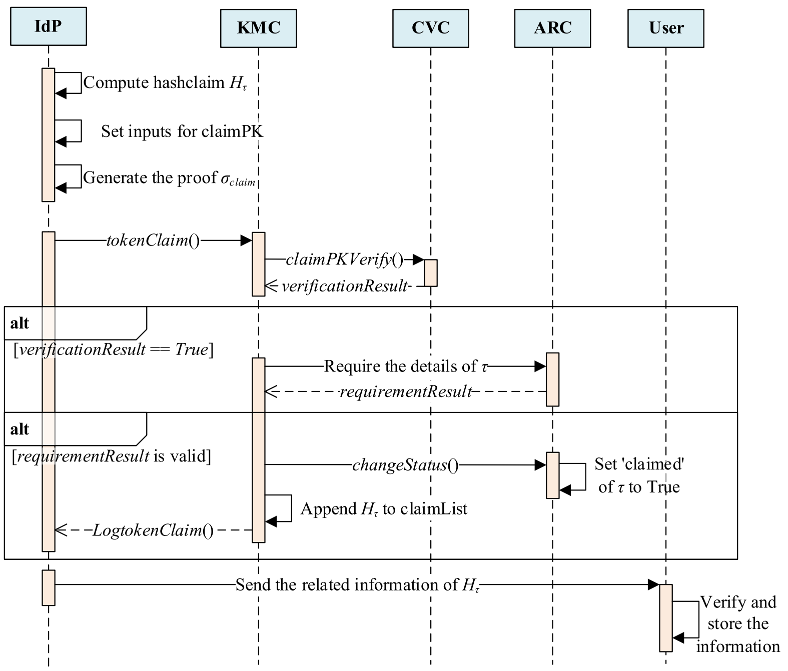

IdP
KMC
CVC
ARC
User
Compute hashclaim $H_\tau$
Set inputs for claimPK
Generate the proof $\sigma_{claim}$
tokenClaim()
claimPKVerify()
verificationResult
alt
[verificationResult == True]
Require the details of $\tau$
requirementResult
alt
[requirementResult is valid]
changeStatus()
Set 'claimed' of $\tau$ to True
Append $H_\tau$ to claimList
LogtokenClaim()
Send the related information of $H_\tau$
Verify and store the information

**Fig. 8 – Sequence diagram of transferring.**

**Algorithm 2 – tokenClaim()**

```
Inputs: $pubInp_{claim}$, $\sigma_{claim}$
Outputs: LogTokenClaim()
  CVC.claimPKVerify($\sigma_{claim}$, $pubInp_{claim}$) $\rightarrow$ verificationResult
  IF verificationResult != True
    Revert
  ELSE
    Require ARC.creatorQuery($\tau$) == msg.sender
    Require ARC.claimedQuery($\tau$) == False
    Require ARC.existQuery($\tau$) == True
    Append $H_\tau$ to the ever-increasing list claimList
    Set the field of 'claimed' of $\tau$ to True
    Emit the event LogTokenClaim(msg.sender,$\tau$,$H_\tau$)
  ENDIF
```

This process successfully transfers the attribute possession to $u$, while ensuring that the possession relation is hidden by zk-SNARK, instead of being public in smart contracts.

## 4.4. Phase 2: responding

### 4.4.1. Particular computation of responseSK

To attest to the possession of the privacy attribute token, the user $u$ needs to prove that he/she owns the ZK private key corresponding to the ZK public key contained in the preimage of hashclaim. In addition, to ensure that proof can only be used once, we design a one-off responding waste, whose preimage includes ZK private key and different response salt figures to distinguish from previous wastes, effectively avoiding replay attacks.

The preimage of $H_\tau$ held by the user is $\{\tau, pk_u^Z, \varepsilon_\tau\}$. We assume that the user's ZK private key is $sk_u^Z$, the 128-bit response salt figure is $\delta$, and the responding waste as the sha256 result of these two values is $N_\delta$. The responseSK can be abstracted to the following formulas:

$pk_u^Z$ equals $sha256(sk_u^Z)$
$H_\tau$ equals $sha256(\tau||pk_u^Z||\varepsilon_\tau)$
$N_\delta$ equals $sha256(sk_u^Z||\delta)$

The $sk_u^Z$, $\delta$ and $\varepsilon_\tau$ are private inputs, $\tau$, $H_\tau$ and $N_\delta$ are public inputs. The Zokrates will perform the following calculations:

a) compute $pk_u^{Z\prime} = sha256(sk_u^Z)$

The newly calculated value $pk_u^{Z\prime}$ should be equal to $pk_u^Z$, but the Zokrates does not need to set $pk_u^Z$ as a private input. The correctness of $sk_u^Z$ will be verified by $pk_u^{Z\prime}$. The correctness of $pk_u^{Z\prime}$ will be verified in step c)

b) compute $H_\tau' := sha256(\tau||pk_u^Z||\varepsilon_\tau)$
c) check $H_\tau' == H_\tau$

The correctness of $\tau$ is verified by the contract. Note that $H_\tau'$ calculated in b) should be equal to $H_\tau$, and the correctness of $H_\tau$ will be verified by the contract. Namely, when feeding the correct ZK private key and token salt figure, the $H_\tau'$ should be equal to $H_\tau$ and exists in the claimList, ensuring the correctness of private inputs $\{sk_u^Z, \varepsilon_\tau\}$.

d) check $sha256(sk_u^Z||\delta)$==$N_\delta$

Step d) guarantees when feeding the correct private input, the $N_\delta$ should be equal to $sha256(sk_u^Z||\delta)$, ensuring the correctness of $N_\delta$.

e) return *True*

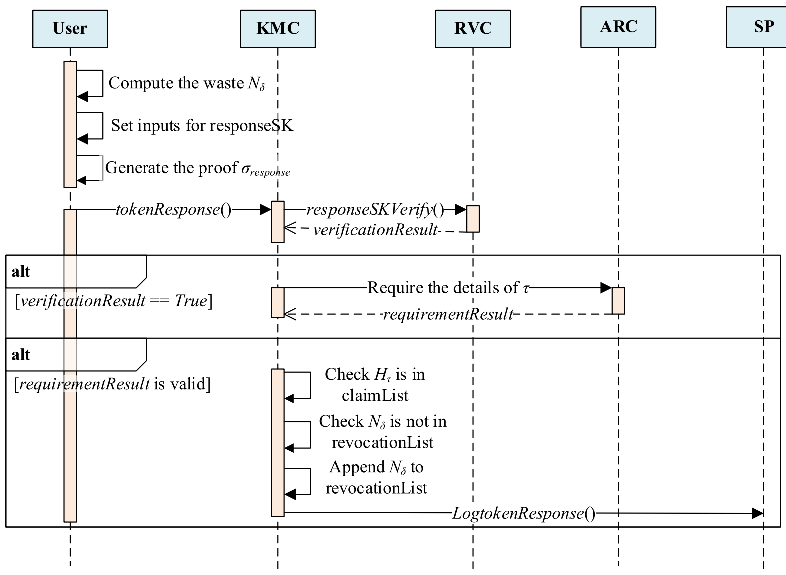

User
KMC
RVC
ARC
SP
Compute the waste $N_\delta$
Set inputs for responseSK
Generate the proof $\sigma_{response}$
tokenResponse()
responseSKVerify()
verificationResult
alt
[verificationResult == True]
Require the details of $\tau$
requirementResult
alt
[requirementResult is valid]
Check $H_\tau$ is in claimList
Check $N_\delta$ is not in revocationList
Append $N_\delta$ to revocationList
LogtokenResponse()

**Fig. 9 – Sequence diagram of responding.**

In this way, the private input is checked against at least one public input in each check phase. By constructing the circuit in this way, the prover can only share the public input and proof which will be verified against the verification key stored in the contract.

### 4.4.2. *The procedure of responding*

The set of constraints of responseSK $Cst_{\text{response}}$ has already been compiled locally. The evidence that can only be decrypted and verified by the SP is called the responding content, whose generation process is explained in 4.6. The detailed procedure of responding is described in Fig. 9.

First, the user $u$ computes the responding waste $N_\delta = sha256(sk_u^Z||\delta)$ and feeds public inputs $pubInp_{\text{response}} = (\tau, H_\tau, N_\delta)$ and private inputs $priInp_{\text{response}} = (sk_u^Z, \varepsilon_\tau, \delta)$ for the circuit of responseSK to generate witness $\psi_{response} = witnessGen(Cst_{\text{response}}, pubInp_{\text{response}}, priInp_{,\text{response}})$. Then $u$ generates the zk-SNARK proof $\sigma_{\text{response}} = proofGen(p_{\text{response}}, \psi_{\text{response}})$ against with the responseSK proving key $p_{\text{response}}$ in Zokrates and calls the *tokenResponse*() in KMC by sending a transaction from anonymous addresses $\Xi_u$ to verify $pubInp_{\text{response}}$ and $\sigma_{\text{response}}$ and enable $ES_\pi$ to be triggered in the event *LogtokenResponse*(). $ES_\pi$ is the attestation of possession of the identity identifier, which is elaborated in 4.6. The details of *tokenResponse*() are shown in Algorithm 3.

The waste will be appended to revocationList to prevent replay attacks. Since the length of the response salt figure is 128 bits, each privacy attribute can perform responding for $2^{128}$ times. It can be considered that users do not have to worry about exhausting the responding times.

**Algorithm 3 – KMC.*tokenResponse*()**

```
Inputs: $pubInp_{\text{response}}$,$\sigma_{\text{response}}$,$ES_\pi$
Outputs: $LogTokenResponse(\tau,ES_\pi)$
  $RVC.responseSKVerifiy(\sigma_{\text{response}}, pubInp_{\text{response}}) \rightarrow verificationResult$
  IF $verificationResult == False$
    revert
  ELSE
    Require $ARC.existQuery(\tau)$==$True$
    Require $ARC.claimedQuery(\tau)$==$True$
    Check $H_\tau$ is already in the list claimList
    Check $N_\delta$ is not in the list revocationList
    Append the $N_\delta$ to the ever-increasing list revocationList
    Emit the event $LogTokenResponse(\tau,ES_\pi)$
  ENDIF
```

**Algorithm 4 – ARC.tokenRevoke()**

```
Inputs: $\tau$
Outputs: $LogTokenRevoke()$
  Require $ARC.creatorQuery(\tau)$==$msg.sender$
  Set the 'exist' of $\tau$ to $False$
  Emit the event $LogTokenRevoke()$
```

## 4.5. *Phase 4: revocation*

As shown in Fig. 10, to revoke the privacy attribute token $\tau$, the IdP call *tokenRevoke*() in ARC from blockchain address $E_{idp}$ recorded in 'creator'. The details of *tokenRevoke*() are described in Algorithm 4. This function will set the 'exist' of $\tau$ to *False* to block functionality of *tokenClaim*() and *tokenResponse*() to complete the revocation of $\tau$.

After watching the event *LogTokenRevoke*(), the IdP notifies the user of the revocation of the $\tau$. Then both parties delete the local attribute information.

## 4.6. *Challenge-response protocol*

The challenge-response protocol allows the user $u$ to demonstrate to the SP that he/she possesses the identity identifier and attributes asserted in the profile without revealing the ZK

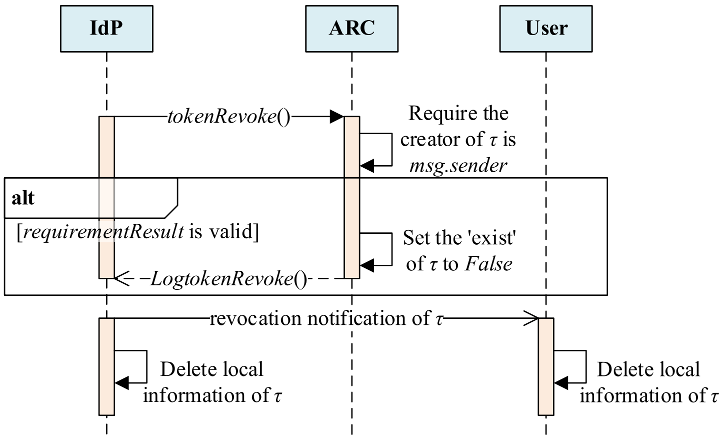

IdP
ARC
User
tokenRevoke()
Require the creator of $\tau$ is msg.sender
alt
[requirementResult is valid]
Set the 'exist' of $\tau$ to False
LogtokenRevoke()
revocation notification of $\tau$
Delete local information of $\tau$
Delete local information of $\tau$

**Fig. 10 – Sequence diagram of revocation.**

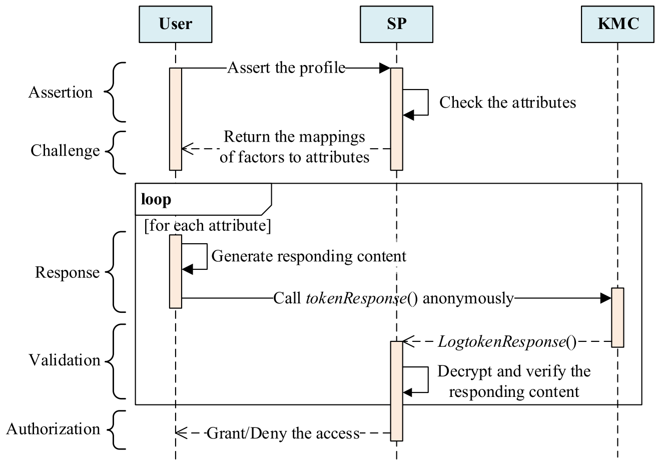


**Fig. 11 – Sequence diagram of the challenge-response protocol.**

private key. However, the sender of the transaction will change the status of the attribute, leading to the exposure of the association between the address and token. Therefore, $u$ should use the one-off anonymous addresses in this protocol.

Before launching the challenge-response protocol, $u$ already knew and possessed the required set of attributes for accessing the service. The sequence diagram of the challenge-response protocol is shown in Fig. 11. And the five components of the protocol are described as follows:

1) Assertion: $u$ sends an assertion of profile packaged in JSON format to the SP through the secure channel. Then SP retrieves and confirms the name-value content and the authority of the creator of each privacy attribute token in ARC.
2) Challenge: For each attribute asserted in the profile, mapping with a 256-bit challenge factor is generated. Then the SP sends these mappings to $u$ through the secure channel and starts watching *LogTokenResponse*() for these attributes.
3) Response: The user needs to compute responding content for each received challenge factor. We assume the challenge factor mapping to $\tau$ is $\pi$. Next, $u$ uses the blockchain private key $sk_u^E$ of address $E_u$ to sign $\pi$ to get the signature $S_\pi$:

$$signatureGen\left(\pi, sk_u^E\right) \rightarrow S_\pi \tag{1}$$

Then $u$ uses the SP's blockchain public key $pk_{sp}^E$ to encrypt the signature $S_\pi$ to get the responding content $ES_\pi$:

$$contentEnc\left(S_\pi, pk_{sp}^E\right) \rightarrow ES_\pi \tag{2}$$

Then $u$ needs to repeat the procedure of responding (elaborated in 4.4) anonymously to trigger *LogTokenResponse*() with each responding content.
4) Validation: The SP watches *LogTokenResponse*() via the event filter and verifies each triggered responding content according to the mappings of factors.

The SP decrypts the responding content contained in *LogTokenResponse*():

$$contentDec\left(ES_\pi, sk_{sp}^E\right) \rightarrow S'_\pi \tag{3}$$

Table 4 – Results of particular computation pairs.

| | Constraints | Key sizes | |
|---|---|---|---|
| | | Proving key(Mbytes) | Verification key(bytes) |
| claimPK | 56985 | 13.4 | 1963 |
| responseSK | 142079 | 39.6 | 2269 |

The SP confirms that $u$ signed the factor with the blockchain private key $sk_u^E$, where *verification Result* $\in$ {*True, False*}:

$$signatureVerify\left(\pi, S_{\pi}^{'}, pk_u^E\right) \rightarrow verificationResult \tag{4}$$

5) Authorization: If all verification results are *True*, the SP performs an authorization operation to grant the access request from $u$.

## 5. Evaluation and analysis

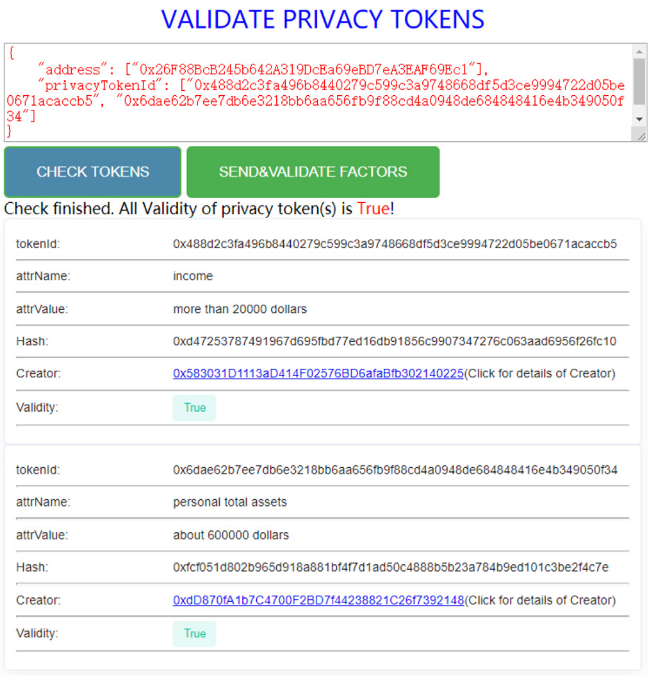
To simulate the application scenario, we developed a prototype which includes smart contracts and a simple client to evaluate the capabilities of BZDIMS. The prototype of smart contracts in BZDIMS was written by solidity, compiled by Remix and uploaded to Github. The client framework is ZoKrates + Python + Web3.js + Html, and the client operating environment is Ubuntu 16.04 (64bit), Intel Core i5-4200@2.5GHz, 4G RAM. The client can generate the zk-SNARK proofs and send transactions to call functions in smart contracts through the ABI, and can also receive and send messages through secure channels.

In the BZDIMS setup phase, the number of computation constraints and key results obtained by compiling two particular computations in BZDIMS are shown in Table 4. The more computation constraints generated, the more complicated the particular computation is, leading to a bigger size of keys.

### 5.1. *Use case*

In this section, we will give an example of a challenge-response protocol use case There is a user named Alice submits a loan application to a bank, and the asserted profile contains Alice's blockchain address and two attribute tokens, one is the income level created by the company and the other is the asset certification created by the government financial department. Fig. 12 shows that the local client of the bank is checking the validity of tokens in profile received from the secure channel. Only when the hash value of the token is correct, 'exist' and 'claimed' of the token are *True*, and the result of 'Validity' will be displayed as *True*. Besides, the creator's authority needs to be further verified according to the access policy.

Then the bank returns the challenge factors and later decrypts and verifies the responding contents in triggered events. Fig. 13 shows that the local client of the bank is decrypting and verifying the responding contents. Only when the decryption factor equals to challenge factor, the result of



```
{
    "address": ["0x26F88BcB245b642A319DcEa69eBD7eA3EAF69Ec1"],
    "privacyTokenId": ["0x488d2c3fa496b8440279c599c3a9748668df5d3ce9994722d05be
0671acaccb5", "0x6dae62b7ee7db6e3218bb6aa656fb9f88cd4a0948de684848416e4b349050f
34"]
}
```

Check finished. All Validity of privacy token(s) is True!

| | |
|---|---|
| tokenId: | 0x488d2c3fa496b8440279c599c3a9748668df5d3ce9994722d05be0671acaccb5 |
| attrName: | income |
| attrValue: | more than 20000 dollars |
| Hash: | 0xd47253787491967d695fbd77ed16db91856c9907347276c063aad6956f26fc10 |
| Creator: | 0x583031D1113aD414F02576BD6afaBfb302140225(Click for details of Creator) |
| Validity: | True |

| | |
|---|---|
| tokenId: | 0x6dae62b7ee7db6e3218bb6aa656fb9f88cd4a0948de684848416e4b349050f34 |
| attrName: | personal total assets |
| attrValue: | about 600000 dollars |
| Hash: | 0xfcf051d802b965d918a881bf4f7d1ad50c4888b5b23a784b9ed101c3be2f4c7e |
| Creator: | 0xdD870fA1b7C4700F2BD7f44238821C26f7392148(Click for details of Creator) |
| Validity: | True |

Fig. 12 – Check tokens in client of SP.

'Factor Validity' will be *True*. Lastly, due to all responding contents are validated correctly, the bank performs service authorization operations.

### 5.2. *Evaluation and analysis*

#### 5.2.1. *Time cost*

The time used to generate witness and proof of two particular computations in the local client is shown in Table 5. And each result is the average of 100 results tested. In general, the time used to generate proof is acceptable under this configuration.

The time used to calculate the zk-SNARK proof depends on the computing resources allocated by the prover, the logic of the code and the amount of computation. In fact, the logic of codes of particular computations is optimal. In addition, if the ZoKrates standard library develops the hash function of Pedersen to replace sha256, the computation required to generate proof will be greatly reduced.

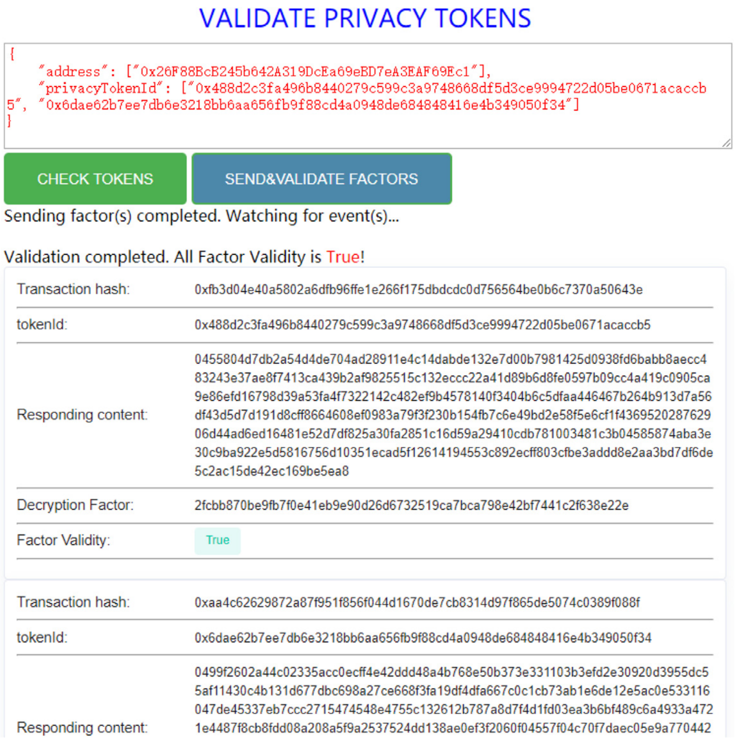
#### 5.2.2. *Cost*

During the analysis, 1 Eth is worth about 131.79USD on December 31, 2019. Since miners can choose transactions with high Gas prices to package, we set $\text{Gas}_{\text{Price}}$ to 5Gwei (0.000000005 Eth / Gas) to ensure that the transaction confirmation time is limited within 15 seconds. The formulas for calculating cost are as follows:

$$\text{Fee}_{\text{Eth}} = \text{Gas}_{\text{Price}} * \text{Gas}_{\text{Used}} \tag{5}$$

$$\text{Fee}_{\text{Fiat}} = \text{Fee}_{\text{Eth}} * \text{Eth}_{\text{Price}} \tag{6}$$

Calling *tokenCreate*() to publish attributes will consume the storage of the blockchain. A token with short data will hence reduce the gas cost. Besides, the length of the IPFS address is



```
{
    ″address″: [″0x26F88BcB245b642A319DcEa69eBD7eA3EAF69Ec1″],
    ″privacyTokenId″: [″0x488d2c3fa496b8440279c599c3a9748668df5d3ce9994722d05be0671acaccb
5″, ″0x6dae62b7ee7db6e3218bb6aa656fb9f88cd4a0948de684848416e4b349050f34″]
}
```

```
Sending factor(s) completed. Watching for event(s)...

Validation completed. All Factor Validity is True!

Transaction hash:     0xfb3d04e40a5802a6dfb96ffe1e266f175dbdcdc0d756564be0b6c7370a50643e
tokenId:              0x488d2c3fa496b8440279c599c3a9748668df5d3ce9994722d05be0671acaccb5
                      0455804d7db2a54d4de704ad28911e4c14dabde132e7d00b7981425d0938fd6babb8aecc4
                      83243e37ae8f7413ca439b2af9825515c132eccc22a41d89b6d8fe0597b09cc4a419c0905ca
                      9e86efd16798d39a53fa4f7322142c482ef9b4578140f3404b6c5dfaa446467b264b913d7a56
Responding content:   df43d5d7d191d8cff8664608ef0983a79f3f230b154fb7c6e49bd2e58f5e6cf1f4369520287629
                      06d44ad6ed16481e52d7df825a30fa2851c16d59a29410cdb781003481c3b04585874aba3e
                      30c9ba922e5d5816756d10351ecad5f12614194553c892ecff803cfbe3addd8e2aa3bd7df6de
                      5c2ac15de42ec169be5ea8
Decryption Factor:    2fcbb870be9fb7f0e41eb9e90d26d6732519ca7bca798e42bf7441c2f638e22e
Factor Validity:      True

Transaction hash:     0xaa4c62629872a87f951f856f044d1670de7cb8314d97f865de5074c0389f088f
tokenId:              0x6dae62b7ee7db6e3218bb6aa656fb9f88cd4a0948de684848416e4b349050f34
                      0499f2602a44c02335acc0ecff4e42ddd48a4b768e50b373e331103b3efd2e30920d3955dc5
                      5af11430c4b131d677dbc698a27ce668f3fa19df4dfa667c0c1cb73ab1e6de12e5ac0e533116
                      047de45337eb7ccc2715474548e4755c132612b787a8d7f4d1fd03ea3b6bf489c6a4933a472
Responding content:   1e4487f8cb8fdd08a208a5f9a2537524dd138ae0ef3f2060f04557f04c70f7daec05e9a770442
```

**Fig. 13 – Validate encrypted factors in responding content in client of SP.**

**Table 5 – Time of generating the proof.**

| | Input Size(Bytes) | | Generation Time(s) | | Total Time(s) |
|---|---|---|---|---|---|
| | Public | Private | Witness | Proof | |
| claimPK | 64 | 32 | 1.861 | 3.332 | 4.193 |
| responseSK | 96 | 64 | 4.579 | 9.717 | 14.296 |

**Table 6 – Cost of calling tokenCreate().**

| Data(bytes) | $Gas_{Used}$ | $Fee_{Eth}$ | $Fee_{Fiat}$(USD) |
|---|---|---|---|
| 100 | 153859 | 0.000769 | 0.101385 |
| 200 | 195061 | 0.000975 | 0.128535 |
| 300 | 215663 | 0.001078 | 0.142111 |
| 400 | 256865 | 0.001284 | 0.169261 |
| IPFS | 153859 | 0.000769 | 0.101385 |

fixed, thus the consumption of off-chain storage of the token is also fixed. The cost of creating the privacy attribute token with different lengths is shown in Table 6.

When performing transferring and responding, the prover needs to send a transaction for proof and public input to be verified in contracts, and the computation required for the EVM of the miner in blockchain to perform the verification will consume massive Gas. In addition, since *tokenRevoke*() changes only one field of the token, so the gas consumption is only 8323. The cost of calling *tokenClaim*(), *tokenResponse*() and *tokenRevoke*() is shown in Table 7.

### 5.2.3. *Throughput*

We expect that the BZDIMS could be deployed in Ethereum, but the confirmation time in public blockchain is a little long and the TPS is only about 14, which greatly limits the application scope of BZDIMS. In fact, since the consensus protocol adopted by the consortium blockchain is more efficient than PoW, the scheme can be deployed in consortium blockchain which is maintained by some organizations and institutions to reduce the confirmation time and increase the throughput. To evaluate the throughput of BZDIMS in the consortium blockchain, the theoretical TPS can be calculated by using the following formula:

$$TPS = \frac{Gas_{Limit}}{Tx_{Gas} * Block_{Time}} \tag{7}$$

Among them, $Gas_{Limit}$ is the gas limit of a single block, $Tx_{Gas}$ is the gas consumption required to execute the transaction, and $Block_{Time}$ is the interval of blocks. If the block interval is too long or the gas limit of a single block is too high, it will lead to a large size of a block and reduce the synchronization speed of the blocks (Schäffer et al. 2019). To avoid affecting the synchronization speed of the block, we set the interval of the block to 5 seconds and the $Gas_{Limit}$ of a single block to 0x2ffffff.

Table 7 – Cost of calling tokenClaim(), tokenResponse() and tokenRevoke().

| | Input Size(Bytes) | | $Gas_{Used}$ | $Fee_{Eth}$ | $Fee_{Fiat}$(USD) |
|---|---|---|---|---|---|
| | Proof | Public Input | | | |
| *tokenClaim()* | 160 | 256 | 294715 | 0.001474 | 0.194202 |
| | 224 | 256 | 304962 | 0.001525 | 0.200955 |
| *tokenResponse()* | | | | | |
| *tokenRevoke()* | - | - | 8323 | 0.000042 | 0.005484 |

Table 8 – Security constraints of BZDIMS.

| Security Constraint | Condition | Purpose |
|---|---|---|
| Zero-knowledge security | An appropriate bilinear group should be chosen | Prevent the inputs used to generate proof from being cracked |
| Particular computation Security | Public input and private input need to be correctly verified. | Prevent users' privacy attribute ownership from being leaked |
| Anti-replay attack | Used proof should not be verified as valid | Prevent attackers to replay the proof to defraud SP and access services |
| Data minimization | When accessing the service, the user does not have to present all the attributes held | Prevent the full identity of the user from being exposed to other entities |
| Identity unlinkability | What attributes the user has should not be known by others | Prevent users' privacy attribute possession from being leaked |
| Behavior privacy | User's access requests should not be redirected to IdP | Prevent user behavior from being leaked by third parties |

*tokenClaim*() and *tokenResponse*() are among the most gas-consuming functions and the throughput of calling them can reach 170TPS and 165TPS respectively in this setting, which can meet most the needs of authentication and application scenarios.

## 5.3. *Security analysis*

The ideal DIMS in blockchain should guarantee the security of the bottom layer and the application layer. At the bottom, all transactions should be executed correctly. At the application layer, the user's privacy cannot be leaked. Therefore, we need to analyze the security constraints as shown in Table 8, which compares the conditions required to implement these security constraints and the purposes of implementing them. The preconditions required for security analysis of the proposed scheme are as follow:

Precondition 1: The SP will not disclose the attribute ownership information asserted in the profile.

Precondition 2: The blockchain private keys and ZK private keys are properly kept and not leaked.

Precondition 3: All preimage information stored in the local database is properly kept and not leaked.

### 5.3.1. *The security of the bottom layer*

**Zero-knowledge security**: Zokrates is based on the zk-SNARK algorithm groth17, and its security has been proven in the original literature (Groth and Maller 2017).

**Particular computation Security**: The formulas are introduced to analyze the security of particular computations in 4.3.1 and 4.4.1, i.e. how to guarantee the correctness of the private inputs and to how to prevent leakage of user privacy when the public inputs are publicly verified.

**Prevention of the replay attack**: The attacker obtains the partial inputs $\{\sigma_{response}, pubInp_{response}\}$of *tokenResponse*() entered by the legitimate user from transactions. The attacker asserts to SP that he/she owns the privacy attribute token $\tau$ and receives a challenge factor $\theta$. Then the attacker signs $\theta$ with his/her blockchain private key and uses the SP's public key to generate a responding content $ES_{\theta}$, which combined with $\{\sigma_{response}, pubInp_{response}\}$ will serve as the attack resource.

Then RVC verifies the proof and the corresponding public inputs according to formula (8):

$$\text{RVC}.responseSKVerifiy(\sigma_{response}, pubInp_{response}) \rightarrow result \quad (8)$$

The returned *result* is *True*, but according to Algorithm 3, *tokenResponse*() detects that the waste $pubInp_{response}$ is in revocationList of KMC, failing to emit *LogTokenResponse*(). Therefore, this scheme can prevent a replay attack.

### 5.3.2. *The security of the application layer*

**Data minimization**: When accessing services from the IdP, the user needs to follow the challenge-response protocol. The profile that contains the necessary attributes to access the service is asserted, instead of all user attributes. Besides, the SP cannot derive all the attributes held by the user based on the user's blockchain address.

**The unlinkability of identity**: If the attacker wants to know the owner of the privacy attribute token, he/she needs to crack the ZK public key contained in the hashclaim. According to the precondition 1 and 3, if the SP is honest and the preimage information stored in the database is not leaked, the attacker can obtain the $pk_u^Z$ and $\varepsilon_\tau$ only through brute force cracking against the formula (9).

$$H_\tau := sha256(\tau || pk_u^Z || \varepsilon_\tau) \quad (9)$$

**Table 9 – Comparisons of blockchain-based DIMSs, where ZKP means Zero-Knowledge Proof and SC means Smart Contract.**

| | Based | Permissionless | Data minimization | Identity unlinkability | Behavior privacy |
|---|---|---|---|---|---|
| Sovrin | ZKP | | √ | √ | |
| uPort | SC | √ | | | |
| SCPKI | SC | √ | | | √ |
| ChainAchor | Hardware | | √ | √ | |
| TCUGA | Graph | √ | | | |
| BZDIMS | SC and ZKP | √ | √ | √ | √ |

The 256-bit $H_\tau$ is the sha256 hash of 256-bit $\tau$, 128-bit $pk_u^Z$, and 128-bit $\varepsilon_\tau$. Among them, the $\tau$ is public input, so the attacker needs to crack a remaining 256-bit preimage. However, using the current computing power to calculate the hash $2^{256}$ times is considered impossible. Therefore, this solution can realize the unlinkability of the user's identity.

**Protection of the behavior privacy of the user**: When the user $u$ performing challenge-response protocol legally, the used one-off anonymous addresses $\Xi_u\{\Xi_u^1, \Xi_u^2, ...\}$ are generated randomly and independently. And the blockchain address $E_u$ of $u$ is independent of the anonymous addresses $\Xi_u\{\Xi_u^1, \Xi_u^2, ...\}$, thus the attacker cannot deduce the blockchain address that represents the user's real identity from the anonymous addresses calling *tokenResponse*().

Besides, the responding content is computed according to formula (1) and formula (2). After the event *LogTokenResponse*() is triggered, the responding content cannot be decrypted by the attacker since the responding content will not be linked to the blockchain address of any user. It can only be decrypted and verified by the SP. Others cannot even know which SP the user is requesting to from the protocol. Therefore, this solution can protect the behavior privacy of the user.

## 6. The related works

The birth of blockchain technology has brought possibilities to solve digital identity puzzles. From industry to academia, all are trying to manage the digital identity from a decentralized perspective.

In industry, the W3C Credentials Community Group designs DIDs (Decentralized Identifiers) (W3C, 2019) to implement the DPKI (decentralized PKI) architecture. DIDs are a new type of identifier for verifiable, decentralized digital identities, and do not rely on any centralized registry so that entities can create and manage their identifiers on any number of distributed, independent roots of trust. DID methods may also be developed for identifiers registered in federated or centralized DIMS.

Besides, companies such as Sovrin (2018) and uPort (Lundkvist et al., 2017) are committed to the application implementations of DIMS. Sovrin is a protocol in the distributed network for self-sovereign identity and decentralized trust, achieving a complete stack to manage the identity from account to the entity so that anyone can issue a verifiable certificate containing a digital signature. Users can selectively expose their identity through zero-knowledge proofs. But in the consensus mechanism PBFT, the existence of the stewards may spawn a group of supernodes acted by large companies, which runs counter to the decentralization. UPort is a secure and self-control DIMS that allows users to act as their own CA. The agent contract address, as the core of uPort, acts as the globally unique persistent identifier. The user uses the private key controlling the agent contract to interact with other smart contracts. This solution allows off-chain storage of attributes (e.g., store JWT signed by the organization on IPFS) that link to uPort identity. Users can submit verifiable proofs for applications, devices or services and revoke them if necessary.

In academia, the SCPKI (Al-Bassam, 2017) implements the idea of The Trust Web through smart contracts, building an example of the claim identity model. Any entity could act as a registrar to sign other people's attributes or revoke the signature. The SCPKI can make it easily possible for rogue certificates to be detected when they are published, but a globally transparent claim model will pose a threat to identity privacy. Azouvi et al. (2017) design a blind register protocol on the blockchain, in which users can publish blind attributes in blockchain and IdP can sign on blind attributes, and users who have collected blind signatures can unblind them locally. This protocol addresses the question of how to register identities and attributes in a system built on globally visible ledgers. The scheme designed by Liu et al. (2017) establishes reputation-based DIMS on the blockchain and ensures the reliability of the identity via smart contract. To build a reputation model, the token is designed to represent the reputation of the entity. To solve the problem of identity and access control in the permissioned blockchain, the ChainAchor (Hardjono and Pentland, 2019) provides an anonymous but verifiable identity for entities. The identity is verified by a verification node built on tamper-resistant hardware that forms a privacy protection layer to allow the entity to selectively disclose the identity during transactions. Lin et al. (2018) proposed a new transitively closed undirected graph authentication (TCUGA), which aims to bind the digital identity to the real-world entity. The TCUGA allows the entity to verify membership via signatures, allowing dynamic addition or removal of vertices and edges in undirected graphs to update existing certificate relationships. Also, the self-sovereign identity scheme proposed by Stokkink and Pouwelse (2018) is adopted by the government, which created a new Dutch digital passport that could prevent identity fraud and provide legal force among citizens. The claim could be verified within a sub-second time.

Table 9 shows the comparisons of some mentioned blockchain-based DIMSs, mainly from the underlying platform and the privacy protection effect. Permissionless blockchain has an absolute decentralization to ensure that transactions can be properly packaged into blocks. All in all, BZDIMS has robust security and effective privacy protection compared with other schemes.

## 7. Conclusion

This paper introduces zk-SNARK into the existing claim identity model to protect identity privacy. The privacy attribute token and two particular computations are designed to realize the secretly transferring the ownership of privacy attributes and attribute ownership authentication. The proposed BZDIMS adopts the framework of off-chain computing and on-chain verifying, which effectively prevents the exposure of the ownership between the user entity and attributes in the distributed ledger, achieving the identity unlinkability and the behavior privacy. Through experimental evaluation and analysis, it is shown that the designed challenge-response protocol can realize low-cost and high-throughput authentication processes, which can be applied to other security mechanisms such as access control and authorization. Future work focuses on overcoming the shortcomings of BZDIMS to make it suitable for a wider range of application scenarios. On the one hand, attributes with the same key-value pair need to be repeatedly created by IdP to distribute to different users. We plan to reduce the redundancy of attributes by optimizing the privacy attribute token logic to reduce the operating cost of BZDIMS. On the other hand, it takes a little long to generate proof off-chain. Perhaps implementing our own ZKP library instead of relying on Zokrates can reduce this time.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRediT authorship contribution statement

**Xiaohui Yang:** Conceptualization, Supervision, Methodology, Funding acquisition. **Wenjie Li:** Data curation, Writing - original draft, Writing - review & editing, Software.

## Acknowledgments

This work is funded by The National Key Research and Development Program of China (no. 2017YFB0802300); and the Post-graduate's Innovation Fund Project of Hebei University (hbu2020ss070).

## REFERENCES

W3C. Decentralized identifiers (DIDs) v0.13 [Internet]. W3C community group final report. 2019 [cited 2019 Aug 30]. Available from: https://w3c-ccg.github.io/did-spec/

Al-Bassam M. SCPKI: a smart contract-based PKI and identity system. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts. ACM; 2017. p. 35–40.

Alpár, G., Hoepman, J.-.H., Siljee, J.. The identity crisis. security, privacy and usability issues in identity management. arXiv Prepr arXiv11010427. 2011;

Androulaki E, Barger A, Bortnikov V, Cachin C, Christidis K, De Caro A, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference. ACM; 2018. p. 30.

Azouvi S, Al-Bassam M, Meiklejohn S. Who am i? Secure identity registration on distributed ledgers.. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology. Springer; 2017. p. 373–89.

Ben-Sasson E, Chiesa A, Tromer E, Virza M. Succinct non-interactive zero knowledge for a von Neumann architecture. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14); 2014. p. 781–96.

Ben-Sasson E, Bentov I, Horesh Y, Riabzev M. Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. 2018;2018:46.

Nakamoto, S. Bitcoin: a peer-to-peer electronic cash system [internet]. Manubot; 2019. Available from: https://git.dhimmel.com/bitcoin-whitepaper/

Castro M, Liskov B. Practical byzantine fault tolerance. In: OSDI; 1999. p. 173–86.

Christensen CM, Hall T, Dillon K, Duncan DS. Know your customers' jobs to be done. Harv. Bus. Rev. 2016;94(9):54–62.

Dunphy P, Petitcolas FAP. A first look at identity management schemes on the blockchain. IEEE Secur. Priv. 2018;16(4):20–9.

Eberhardt J, Tai S. ZoKrates-scalable privacy-preserving off-chain computations. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE; 2018. p. 1084–91.

Efanov D, Roschin P. The all-pervasiveness of the blockchain technology. Procedia Comput. Sci. 2018;123:116–21.

Ferdous MS, Norman G, Poet R. Mathematical modelling of identity, identity management and other related topics. In: Proceedings of the 7th International Conference on Security of Information and Networks; 2014. p. 9–16.

Ferdous MS, Chowdhury F, Alassafi MO. In search of self-sovereign identity leveraging blockchain technology. IEEE Access 2019;7:103059–79.

Gaetani, E., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., Sassone, V.. Blockchain-based database to ensure data integrity in cloud computing environments. 2017;

Giacomelli I, Madsen J, Orlandi C. Zkboo: Faster zero-knowledge for boolean circuits. In: 25th {USENIX} Security Symposium ({USENIX} Security 16); 2016. p. 1069–83.

Groth J, Maller M. In: Annual International Cryptology Conference. Snarky signatures : minimal signatures of knowledge from; 2017.

Hardjono, T., Pentland, A. Verifiable anonymous identities and access control in permissioned blockchains. arXiv Prepr arXiv190304584. 2019;

Ingram, D.. Facebook says data leak hits 87 million users, widening privacy scandal [internet]. Reuters. 2018 [cited 2019 Sep 3]. Available from: https://www.reuters.com/article/us-facebook-privacy/

facebook-says-data-leak-hits-87-million-users-widening-privacy-scandal-idUSKCN1HB2CM

Konda, C., Michael Connor, D.W., Quentin Drouot, P.B.. Nightfall [internet]. EY global blockchain R&D. 2019. Available from: https://github.com/EYBlockchain/nightfall/blob/master/doc/whitepaper/nightfall-v1.pdf

Lesavre, L., Varin, P., Mell, P., Davidson, M., Shook, J.. A taxonomic approach to understanding emerging blockchain identity management systems. arXiv Prepr arXiv190800929. 2019;

Lin C, He D, Huang X, Khurram Khan M, Choo KKR. A new transitively closed undirected graph authentication scheme for blockchain-based identity management systems. IEEE Access 2018;6:28203–12.

Liu Y, Zhao Z, Guo G, Wang X, Tan Z, Wang S. An identity management system based on blockchain. In: 2017 15th Annual Conference on Privacy, Security and Trust (PST). IEEE; 2017. p. 44–4409.

Lundkvist, C., Heck, R., Torstensson, J., Mitton, Z., Sena, M.. Uport: a platform for self-sovereign identity. URL: https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf. 2017.

Mingxiao D, Xiaofeng M, Zhe Z, Xiangwei W, Qijun C. A review on consensus algorithm of blockchain. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC); 2017. p. 2567–72.

Morais E, Koens T, van Wijk C, Koren A. A survey on zero knowledge range proofs and applications. SN Appl. Sci. 2019;1(8):1–17.

Schäffer M, di Angelo M, Salzer G. Performance and scalability of private ethereum blockchains. In: International Conference on Business Process Management. Springer; 2019. p. 103–18.

Sovrin. Sovrin: a protocol and token for self-sovereign identity and decentralized trust [internet]. Sovrin. 2018. Available from: https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf

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). IEEE; 2018. p. 1336–42.

Wei, J.. Huazhu hotels group investigates alleged info leak [Internet]. ChinaDaily. 2018 [cited 2019 Sep 4]. Available from: http://www.chinadaily.com.cn/a/201808/29/WS5b86473da310add14f38871b.html

Wood G. Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. 2014;151(2014):1–32.

Zhu X, Badr Y. Identity management systems for the internet of things: a survey towards blockchain solutions. Sensors 2018;18(12):4215.

Zikratov I, Kuzmin A, Akimenko V, Niculichev V, Yalansky L. Ensuring data integrity using blockchain technology. In: 2017 20th Conference of Open Innovations Association (FRUCT). IEEE; 2017. p. 534–9.

**Xiaohui Yang** is a Professor at the School of Cyber Security and Computer, Hebei University, China. He received his PhD degree from the University of Science and Technology of China in 2010. He has published several research papers in reputed international journals and conferences. His primary research interests include distributed computing, information security and trusted computing.

**Wenjie Li** is currently a master candidate in the School of Cyber Security and Computer, Hebei University of China where he is working on Cryptography and Blockchain technology novel applications, such as blockchain-based systems, decentralized applications, smart contracts programming and security and software development. His major research interests include information security, peer-to-peer networks, cryptocurrencies and blockchains.