Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Privacy-Preserving Blockchain-Based Healthcare System for IoT Devices using zk-SNARK

DUC ANH LUONG[1], JONG HWAN PARK[2]

[1]Department of Computer Science, Sangmyung University, Seoul 03375, South Korea.
[2]Department of Computer Science, Sangmyung University, Seoul, South Korea.

Corresponding author: Duc Anh Luong (adluong11@gmail.com).

This work was supported by Sangmyung university.

ABSTRACT Privacy is playing a crucial role in the smart health industry, where health service providers and their customers use the internet of things (IoT) to provide and consume health services. Preserving privacy for legitimate users and preventing illegitimate users from accessing services are difficult to implement simultaneously. In this study, we addressed this issue by proposing a new healthcare system for IoT based on the blockchain and zero-knowledge succinct noninteractive argument of knowledge (zk-SNARK). We employ the anonymity property of the public blockchain to protect users' privacy. The zk-SNARK scheme works as an anonymous authenticator to prevent unauthorized users from using services. We also analyze the security of the proposed system by showing that it can resist various types of attacks, such as impersonation, collusion, and man-in-the-middle attacks. Finally, we evaluate the performance of the zk-SNARK scheme with respect to computational costs and the interactions with the Ethereum blockchain smart contract with respect to transaction fees.

INDEX TERMS blockchain, healthcare, IoT, privacy, zk-SNARK, anonymous authentication.

I. INTRODUCTION

Environmental pollutions cause diseases and adversely affect people. To prevent diseases and improve quality of life, healthcare is necessary and should be provided to all people who need it, thereby tremendously increasing patients seeking medical care, and it is becoming difficult to access caregivers or physicians directly and physically. The advent of the internet of things (IoT) or IoT health devices that can track patients' health conditions and health status has improved healthcare quality by allowing patients to send their health data to physicians through the internet. Basically, these IoT health devices can be divided into two: high- and low-level devices.

- High-level devices: collect health data from patients and send them directly to physicians through the internet. These devices can be wearable devices such as smartwatches, smartphones, or stationary devices.
- Low-level devices: also collect health data from patients but cannot send them directly to physicians through the internet. These devices need a medium device such as a smartphone or computer to gather and format collected data and send them to physicians. They are medical embedded devices such as ingestible sensors or connected inhalers.

The process of collecting health data from patients and transmitting them via the internet to physicians or a health service provider (HSP) is called remote patient monitoring (RPM) [1].

In practice, data are not sent directly from patients to physicians but rather stored in cloud storage where physicians work (the HSP). By accessing this cloud, they can process and analyze their patients' health data and provide treatments or health alerts to the patients through IoT devices. Fig. 1 describes this process.

Cloud storage could be provided by a third party or the HSP. Either ways, data need to be securely transmitted between the cloud server and IoT devices. Because the communication between the cloud server and IoT devices is wireless, it is vulnerable to several attacks. For example, malicious adversaries can eavesdrop on messages to obtain sensitive information about patients. In worse cases, they can tamper or change data to break the service and make the

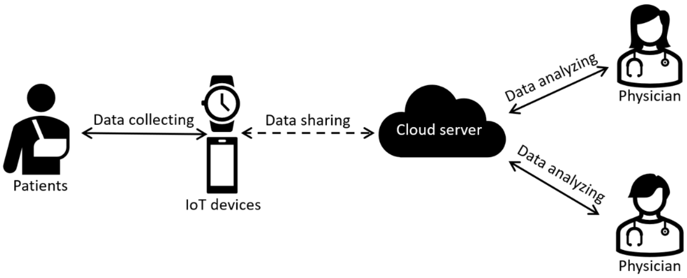


FIGURE 1: Data sharing between health IoT devices and physicians

physicians give wrong treatments. Therefore, a data-sharing process must be protected to ensure confidentiality, integrity, and availability (also known as the CIA triangle).

A. MOTIVATION

Besides the CIA triangle, privacy plays a crucial role in a health data-sharing system. Traditionally, people look for health services when they have health problems. Using these services, they must provide their identities, and physicians will give them treatments or health alerts after analyzing their information, such as symptoms, health conditions, and medical history. However, when people care more about their quality of life in modern life, they may look for healthcare services even in good health conditions to ensure that their health does not have potential harm. These services allow physicians to track their daily health status and other sensitive information to ensure they are always in good shape and alert them when something abnormal is observed. In these cases, physicians only need to know essential details of their health conditions and medical history; thus, people should be able to hide their identities. Naturally, besides efficiency, a healthcare system needs the following security requirements.

1) Only legitimate users, who paid for healthcare services, should be able to use health services from HSPs.
2) Health data between IoT devices and the HSP need to be protected from malicious adversaries using appropriate encryption schemes.
3) Privacy should be protected so that the HSP and adversaries do not know who the health data owner is.

The first and second requirements can be solved by combining a standard authentication technique with an encryption scheme, whereas the third requirement is not easy to be handled simultaneously with the first one. Most existing healthcare systems use a trusted third party to authenticate users. This solution can hide users' identities from the HSP and adversaries. However, using a trusted third party is not a good solution because the users' privacy relies on this party. There is nothing to guarantee that the trusted third party will not collude with the HSP to disclose users' identities and trace their health data.

The advent of the blockchain provided a better solution, thanks to its anonymity property. The HSP can sell its services in a public blockchain network, where users are free to join and quit. For the first requirement, users can easily generate a blockchain address to buy health services, and because their addresses are unique, the HSP can use them to identify legitimate users. The third requirement is also satisfied because the public blockchain allows users to create an address without providing their real identities.

However, there are cases when the HSP needs to classify users, for instance, splitting users into two groups based on whether users have health insurance. Users with insurance should have the services at a lower price than non-insurance users. In these cases, using the pure public blockchain to conceal users' identities is inefficient. Because users' addresses are anonymous, the HSP cannot classify users without information about their identities. One solution to address this issue is blockchain-based anonymous authentication. Using this technique, users must show their identities to the HSP when signing up for the health services, so that the HSP can classify them first. Later, their identities can be hidden using their blockchain address in the anonymous authentication phase.

One of the anonymous authentication techniques is zero-knowledge proof (ZKP) [2], by which a prover can convince a verifier that he knows a secret witness without revealing it. Applying the ZKP technique to a healthcare system, a user (as a prover) can convince the HSP (as a verifier) that he is a registered user who passed some predefined restriction conditions without showing the user's identity (as a witness) to the HSP. Therefore, users can protect their privacy when using health services from the HSP, whereas the HSP can still classify their users and guarantee that they provide health services for legitimate users.

Nevertheless, anonymous authentication using the ZKP technique has one issue; anyone with a correct witness can generate the correct proof and pass the authentication phase. This issue results in collusion attacks in which attackers have access to users' secret witnesses or malicious users intentionally leak their secret witnesses so that unauthorized users can use this witness to generate valid proof and get authenticated in the anonymous authentication phase. It is difficult to counter this attack scenario and protect users' privacy simultaneously because (1) the user's secret witnesses are hidden using the ZKP technique in the anonymous authentication phase and (2) the ZKP is generated differently even if the same witness is used. This issue leads to a financial loss for the HSP when unregistered users can use their services without being detected. Therefore, a new privacy-preserving blockchain-based healthcare system for IoT, which provides collusion attack resistance without trusted third parties, should be designed.

B. CONTRIBUTIONS

The major contributions of this study are as follows.

- We propose a privacy-preserving healthcare system that securely shares health data between IoT devices and the HSP. The proposed system is based on a simple and efficient anonymous authentication scheme using the zero-knowledge succinct noninteractive argument of knowledge (zk-SNARK) [3] (as the ZKP) in combination with the public blockchain.
- The proposed system also provides the authentication property against collusion attacks, without relying on a trusted third party. In the anonymous authentication phase, the zk-SNARK scheme is designed to detect if the same witness is used more than once.
- We present security requirements for a healthcare system and analyze the proposed system's security based on these requirements. We also give implementation results simulated on the Ropsten test network using the Zokrates toolbox [4]. Our implementation focuses on the performance of a smart contract (SC) working with the designed zk-SNARK scheme and evaluates its relevant computational costs and transaction fees based on the number of users.

II. RELATED WORKS

To date, several works have been proposed to design health data sharing systems. In this section, we briefly review a few systems that relate to our work.

Dwivedi *et al.* [5] suggested a decentralized privacy-preserving healthcare blockchain system for IoT devices, where user anonymity is achieved using a ring signature scheme [11]. In their system, health data are encrypted under a symmetric-key encryption (SKE) scheme and signed by the ring signature so that the HSP (as a verifier) can ensure that the data sender belongs to a valid group of users but cannot identify exactly which user it is. Health data are not directly sent from the users to the HSP. Instead, they are sent to the blockchain (overlay in their term). After verifying that the sender is authenticated by an authority, the blockchain network sends these data to the HSP. However, all users in this system must be authenticated by a trusted authority. If the authority colludes with the HSP, the HSP can reveal the owners of all health data. Badr *et al.* [6] introduced a multitier blockchain framework for IoT-electronic health record systems in which users generate their pseudonyms on a blockchain network and use them to communicate with the HSP to protect their identities. However, user anonymity also relies on the trust of a public authority. Fu *et al.* [7] introduced a healthcare blockchain system based on lightweight message-sharing. The interleaving encoder encrypts health data as an electronic medical record into n shares, and these shares are stored in different local blockchain nodes. However, the system does not provide user anonymity to the HSP, which can decode and recover entire health data. A similar drawback can also be found in other constructions [12], [13] that allow the HSP to recover health data and their relevant information on a user. Griggs *et al.* [8] suggested a healthcare blockchain system using SCs for a secure automated RPM system. In this system, health data from users are collected by IoT devices and sent directly to an SC in plaintext. The SC analyzes these data and sends alerts to the users and HSP. This system uses the anonymity property of the public blockchain to protect users' privacy. However, to receive the service, users must use their real identities to register their IoT devices with the HSP and have permission to use the SC from the HSP, meaning that user anonymity is not provided to the HSP.

Yin *et al.* [9] introduced an IoT-based anonymous function for security and privacy in healthcare sensor networks. Users in this system encrypt their health data and send them to a data center. The anonymization process is triggered when the data center sends users' data to the HSP or other organizations. However, this system is centralized so that users' data can be collected and manipulated in the data center before being sent to other organizations. Attarian and Hashemi [10] presented an anonymity communication protocol for health data-sharing, where user anonymity is guaranteed using ring signature and noninteractive ZKP techniques [14] proof techniques. Then, health data are encrypted and sent through three intermediate nodes in a peer-to-peer network before reaching the target HSP node. However, before participating in their healthcare network, all participants must be authenticated by the so-called healthcare authority, which has the right to explore the reality of participants' identities. This means that if the healthcare authority is malicious, user anonymity could be broken.

Based on the above literature review, some systems do not have collusion attack-resistance property, whereas others rely on a trusted third party (certification authority) to counter collusion attacks and prevent unauthorized users from joining the system. If the certification authority is malicious and colludes with the HSP, user anonymity is no longer provided. We aim to present a new blockchain-based privacy-preserving healthcare system for IoT devices to address the above issues.

III. PRELIMINARIES

In this section, we briefly describe blockchain and cryptographic schemes used in the proposed system.

A. BLOCKCHAIN

A blockchain is a distributed database shared via cryptography among the nodes of a computer network. The blockchain concept was first introduced by Satoshi Nakamoto in 2008 in the cryptocurrency bitcoin project [15]. Blockchains are typically managed in a peer-to-peer network using a consensus algorithm. Currently, proof of work (PoW) [16] and proof of stake (PoS) [17] are two of the primary consensus algorithms. Basically, there are two types of blockchains: public and private blockchains. In the former, data in blocks are transparent to everyone, and people are free to join and leave the system. Bitcoin and Ethereum [18] are the two most popular public blockchain

Authors	Privacy techniques	Trusted third party	User privacy	Collusion attack resistance
Dwivedi *et al.* [5]	Ring signature	Yes	Fair	No
Badr *et al.* [6]	Pseudonym-based encryption	Yes	Low	No
Fu *et al.* [7]	Secret sharing	Yes	Low	No
Griggs *et al.* [8]	Public blockchain	Yes	Low	Yes
Yin *et al.* [9]	Homomorphic encryption	Yes	Low	Yes
Attarian and Hashemi [10]	Ring signature	Yes	Fair	Yes
Our system	zk-SNARK	No	High	Yes

TABLE 1: System comparison

systems. For the latter, data are transparent to only system members, and only people with permission can join. The private blockchain is not fully decentralized, and some parties have more power than other nodes. Commonly, some parties have the right to allow nodes to join a network, or they can be validators who create data blocks and commit to the blockchain. Hyperledger Sawtooth [19], and Hyperledger Fabric [20] are typical private blockchain systems. Below are the most crucial properties of a blockchain.

- Immutability: once data are stored in a blockchain, it is infeasible to change or remove them. Data can only be updated by adding a new block, but the old version is still maintained in the chain.
- Decentralized: no entity entirely governs the blockchain network. Data are recorded to the chain by miners or validators if the transaction satisfies the network's policy and is not tampered with by a malicious node. No one in the network can personally add a block to the chain or deny a valid block to be added.
- Consensus: a block is considered valid if and only if more than half the number of nodes in the network agree with its validity. The chain of these valid blocks is the main chain. Blocks that are not in the main chain are orphan blocks. Orphan blocks are correctly minted, and the data in them are corrected.

In a blockchain network, nodes communicate by sending and receiving transactions. A transaction is a data package containing information about the sender, receiver, and content the sender wants to send to the receiver. Nodes can send transactions to other nodes or SCs. An SC is simply a program stored on a blockchain network that runs when predetermined conditions are met. It can be triggered by transactions received from nodes or other SCs. Because the SC is a type of data on the blockchain, once SCs are deployed to the blockchain network, they are immutable and transparent to all nodes in the blockchain network.

B. ZK-SNARK

zk-SNARK scheme [3], [21], [22] is a type of ZKP that allows a prover to convince a verifier that the prover knows a witness without revealing any information about that witness. It is presented by four algorithms as follows: Setup, Keygen, Proof, and Verify.

- $\mathsf{Setup}(1^\lambda) \rightarrow Z$: given a security parameter λ, a set of public parameters $Z = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, g_1, g_2, e)$ is generated, where p is a prime number, $\mathbb{G}_1$, and $\mathbb{G}_2$ are two cyclic groups of order p with generators g_1, and g_2, respectively, e is a bilinear map [23] so that $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$.
- $\mathsf{KeyGen}(C, Z) \rightarrow (\mathsf{PK}, \mathsf{VK})$: the key generation algorithm takes an arithmetic circuit C and Z as inputs, and after transforming C into polynomials and a relation R_C, the algorithm generates a proving key PK and a verification key VK.
- $\mathsf{Proof}(\mathsf{PK}, \vec{x}, m) \rightarrow \pi$: the proof generation algorithm takes as inputs the public key PK, a public statement $\vec{x}$ (as a public input of the circuit C), and a secret witness m of the prover. The algorithm outputs a zero-knowledge proof π with respect to the relation of $\vec{x}$ and m through the circuit C.
- $\mathsf{Verify}(\mathsf{VK}, \vec{x}, \pi) \rightarrow b$: the verification algorithm takes VK, $\vec{x}$, and π as inputs, and outputs a binary bit b indicating whether the proof π is valid $(b = 1)$ or invalid $(b = 0)$.

In terms of security requirements, the zk-SNARK scheme has the following properties:

- *Completeness*: the prover can generate a correct proof if he has the witness m. Then, the verifier always accepts the correct proof.
- *Zero-knowledge*: the prover can prove that he has the secret m without revealing any information on m to the verifier (and others).
- *Soundness*: without the proper witness m, it is infeasible for a cheating prover to generate correct proof.

C. DIGITAL SIGNATURE

A digital signature scheme comprises the following three algorithms: S_keygen, S_sign, and S_verify.

- $\mathsf{S\_keygen}(1^\lambda) \rightarrow (\mathsf{PK}, \mathsf{SK})$: given a security parameter λ, the key generation algorithm outputs a pair of keys $(\mathsf{PK}, \mathsf{SK})$, where PK and SK are a public key and signing key, respectively.
- $\mathsf{S\_sign}(M, \mathsf{SK}) \rightarrow \sigma$: the signature generation algorithm takes as input a message M, SK; and outputs a signature σ.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3177211, IEEE Access

- S_verify$(M, \sigma, \mathsf{PK}) \to 0/1$: the verification algorithm takes as inputs a message M, a signature σ, and PK; and outputs a bit b, which indicates that σ is valid $(b = 1)$ and invalid $(b = 0)$.

As the security of a digital signature scheme, we consider the unforgeability against chosen message attacks [24], meaning that it is infeasible for an adversary to generate a forged signature on a message m without a signing key of a signer.

D. HASH FUNCTION

We consider a hash function $H\{0,1\}^* \to \{0,1\}^\ell$, where ℓ is a positive integer and is determined by a security parameter λ, which satisfies the following property:

- *Collision-resistance:* it is infeasible for an adversary to find two preimages m_1, and m_2 such that $H(m_1) = H(m_2)$.

In addition, we employ the fact that the collision-resistant property implies the onewayness and second preimage resistance properties.

E. DIFFIE-HELLMAN KEY EXCHANGE (DHKE)

Given a DHKE parameter (g, p), where p is a prime number and g is a generator, two parties A and B wishing to share a secret key K do so as follows.

- A sends B $M_A = g^a \pmod{p}$ for a random $a \in \mathbb{Z}_p$.
- B sends A $M_B = g^b \pmod{p}$ for a random $b \in \mathbb{Z}_p$.
- A computes $K = (M_B)^a \pmod{p}$ and B computes $K = (M_A)^b \pmod{p}$.

In our construction, the well-known man-in-the-middle attack is protected by signing M_A and M_B under the signing keys of A and B, respectively.

IV. PROPOSED SYSTEM

A. MAIN ENTITIES

In the proposed system, there are six main entities: users, IoT devices, pseudonyms, an SC, the HSP, and the public address of the HSP on a blockchain network. Their relations are depicted in Fig. 2.

- *User (U):* we can have many users in this system as the consumers of health services. Users employ health services from the HSP.
- *Pseudonym (PS):* this is a user's public address generated on the blockchain network responsible for calling SC's functions. Notably, a signing key corresponding to a pseudonym will be used for signing a transaction to an SC. There are also many pseudonyms corresponding to a user in this system. Pseudonyms do not link to or disclose users' identities and are necessary for anonymously registering IoT devices to the SC.
- *IoT devices:* these are wearable devices or other health data-collecting devices. One user may have many IoT devices. These devices collect health data from the user and share them with the HSP.
- *SC:* this contains the zk-SNARK verification function (using pseudonyms received from users) and its relevant verification key. The SC additionally contains IoT device parameters during the registration of IoT devices. As the number of pseudonyms increases, the number of SCs can also increase.
- *HSP:* there is one HSP in the proposed system. The HSP is responsible for building SC, collecting health data from users through IoT devices, storing these data in its data center, and analyzing it to provide users with health services.
- *HSP's blockchain address:* this is responsible for deploying SC to the blockchain network and querying for devices' parameters.

As shown in Fig. 2, there are four channels in the proposed system. (1) The blockchain network is the blockchain channel where data are shared in the form of transactions. (2) The insecure channel is a data-sharing channel on the internet where data are unencrypted. (3) The secure channel is a data-sharing channel on the internet in which data are encrypted using symmetric-key ciphers. (4) The secure local channel is a local channel not using the internet.

B. SYSTEM DESCRIPTION

For ease of reading, Table 2 presents notations used in the proposed system. In addition, we assume the following.

TABLE 2: Notations in the proposed system

Notations	Descriptions
SK_{sig}	Signing key of HSP
PK_{sig}	Verification key of HSP
PK	zk-SNARK proving key
VK	zk-SNARK verification key
H	Hash function
SC	Smart contract
AD	Address of smart contract
U_i	i^{th} user
$\vec{x}_i$	Public statement of i^{th} user
ID_i	Device identity of i^{th} user
M_i	Token of pseudonym generated by i^th user
m_i	Preimage of i^{th} user
PS_i	Pseudonym of i^{th} user

- The HSP generated a pair of keys $(\mathsf{PK}_{sig}, \mathsf{SK}_{sig})$, and all users have PK_{sig}. The digital signature scheme is ECDSA.
- U_i owns an identity ID_i corresponding to an IoT device, which does not reveal U_i's identity.
- ID_i has built-in functions to perform the DHKE and securely communicate with the HSP (using symmetric-key ciphers). ID_i can receive data from the HSP and show them to U_i.
- All inputs to SC are generated as transactions on the blockchain network, meaning that each transaction

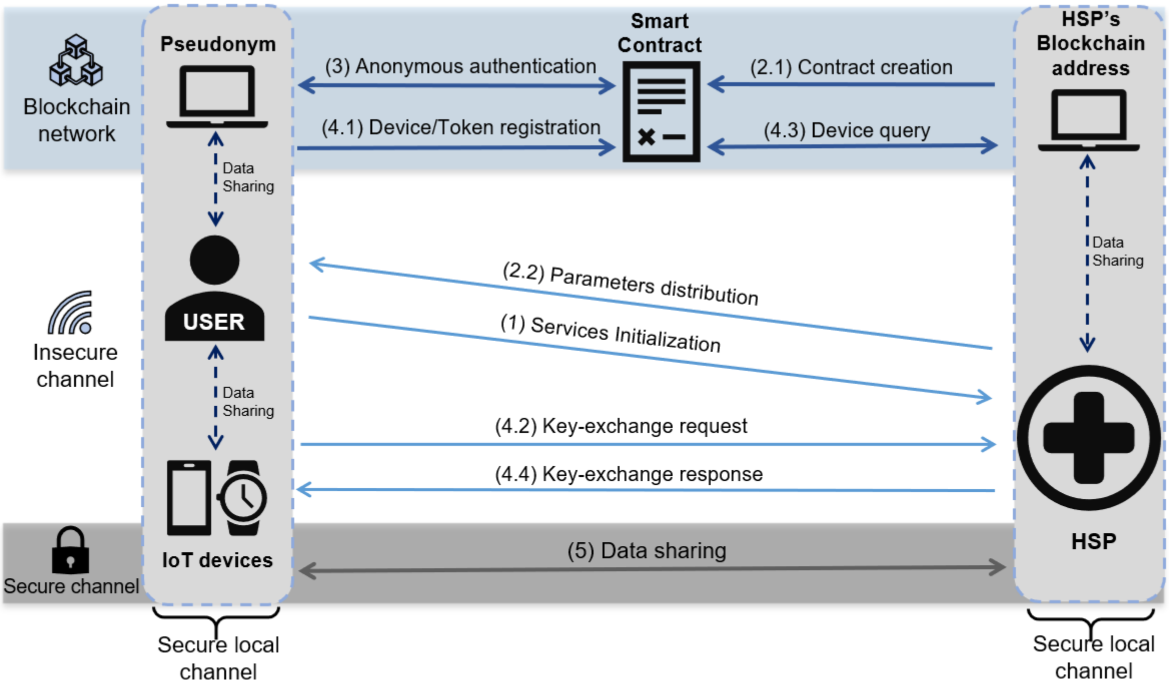


FIGURE 2: System model

should be signed by the owner of a transaction and can be publicly verified, especially by miners when generating a block.

- The secure local channels are kept secure and attack-resistant using a separate method (which is out of the scope of this study).
- The DHKE parameters (p, g) are shared in advance by all entities.

With Fig. 2, we now describe our system as follows:

Step 1: Service initialization

- U_i creates a public address on the blockchain network as a pseudonym. The address must not disclose any information about U_i's identity so that attackers cannot trace the U_i's identity from this address.
- U_i selects a random $m_i \in \{0,1\}^\ell$ for some $\ell \in \mathbb{Z}^+$, where ℓ is determined by the security parameter λ, and computes $H(m_i) = h_i$. Afterward, U_i sends h_i to the HSP and keeps m_i secure as a preimage of h_i.
- The HSP collects hash values $\{h_i\}$ from users. Based on the $\{h_i\}$, the HSP generates an arithmetic circuit C. We emphasize that the number of collected h_i values is critical to the performance of the zk-SNARK Setup and Proof algorithms. In Section VI, we show the performance results of such algorithms, depending on various numbers of h_i values. The detail of circuit C is described in Algorithm 1.

Step 2: SC building and parameter distribution

- The HSP generates a pair of zk-SNARK proving and verifying keys (PK, VK) based on the circuit C, such as $\mathsf{Setup}(1^\lambda) \to Z$ and $\mathsf{KeyGen}(C, Z) \to (\mathsf{PK}, \mathsf{VK})$.

Algorithm 1: Arithmetic circuit C

```
/* This program will check whether its
   (private) input $m_i$ can be hashed to one of
   the pre-defined hash outputs $\{h_i\}_i^n$ for
   some positive integer $n$. If so, the
   program also checks if its (public) input
   $\hat{h}_i$ is equal to the hash value $H(m_i+1)$. If
   both conditions are satisfied then return
   1, else return 0.                          */
Function main(private uint $m_i$, public uint $\hat{h}_i$)
/* List of hash values                        */
uint[$n$] user;
user[0] = 0x123..34;
user[1] = 0x234...54;
...
user[$n$] = 0x432...32;
/* hash function                              */
$h_i \leftarrow H(m_i)$;
/* check if h is on the list of hash values
   */
bool state = false
uint $r$ = 0
for $i = 0; i <= n; i++$ do
    if $h_i$==user[i] then state = true;
end
if (state && $\hat{h}_i$ == $H(m_i+1)$) then r = 1;
return r
```

- The HSP creates an SC containing (1) the zk-SNARK verification function using the VK, (2) a function that allows a user to add (or remove) its own device's identity, and (3) a function for the HSP to query to check the validity of a device's identity. Algorithm 2 describes SC in detail. Afterward, the HSP uses its public address to deploy SC to the blockchain network. This step corresponds to *(2.1) Contract creation* in Fig. 2.

- The HSP creates a data package $P = (AD, \mathsf{PK})$ and generates a signature $\sigma = \mathsf{S}\_sign(P, \mathsf{SK}_{sig})$ using its signing key SK_{sig}. (P, σ) is sent to all users who sent those $\{h_i\}$ values. This corresponds to *(2.2) Parameters distribution* in Fig. 2.
- When receiving (P, σ), U_i verifies the signature using PK_{sig} of the HSP. If the signature is valid, U_i proceeds to the next step.

Step 3: Anonymous authentication between PS_i and SC

This step corresponds to *(3) Anonymous authentication* in Fig. 2. After verifying that $P = (AD, \mathsf{PK})$ is from the HSP, U_i generates the zk-SNARK proof π_i (regarding m_i) and asks SC for authentication as follows.

- U_i computes $\mathsf{Proof}(\mathsf{PK}, \vec{x}_i, m_i) \rightarrow \pi_i$. Notably, $\vec{x}_i$ is an array of circuit C's public input and output, which is uniquely determined by m_i. As in Algorithm 1, because the circuit C has both public input and output, $\vec{x}_i$ comprises the public input $\hat{h}_i \leftarrow H(m_i + 1)$, and r. That is, $\vec{x}_i = \{\hat{h}_i, r\}$.
- U_i uses PS_i to interact with SC, and invokes the function **verifyTx()** with π_i and $\vec{x}_i$. As usual, note that the transaction for **verifyTx(**π_i**,** $\vec{x}_i$**)** is verified under PS_i (and its relevant public key).
- The function **verifyTx()** is implemented as the zk-SNARK Verify algorithm. If m_i is one of preimages with respect to those hash values listed in the arithmetic circuit C and $\vec{x}_i$ has not been used yet, then the function **verifyTx(**$\pi_i, \vec{x}_i$**)** returns $true$. In that case, SC authenticates PS_i by adding $(PS_i, \vec{x}_i)$ into a table of authenticated pseudonyms. We see that SC is programmed to register $\vec{x}_i$ only once with respect to the corresponding h_i.

Step 4: Authentication and key exchange between IoT devices and the HSP

This step corresponds to (4.1), (4.2), (4.3), and (4.4) in Fig. 2.

- U_i selects a random $a \in \mathbb{Z}_p$ and computes $M_i = g^a \pmod p$. M_i is the public token of PS_i, which will be used for the DHKE.
- To register ID_i for an IoT device, U_i uses PS_i to register a tuple of data $T_i = (M_i, ID_i)$ to SC by calling the **addDevices(**T_i**)** function. The transaction for **addDevices(**T_i**)** should be verified under PS_i (and its relevant public key). If the verification succeeds, SC stores $(PS_i, \vec{x}_i, M_i, ID_i)$ into a table of authenticated devices as in Table 3. This step corresponds to *(4.1) Device/Token registration* in Fig. 2.
  As shown in Table 3, each PS_i is associated with its corresponding $(\vec{x}_i, M_i, ID_i)$, which allows the HSP to perform the DHKE protocol only with those IoT devices registered in this table.

Algorithm 2: The smart contract

```
Structure Pseudonym
    uint256 $address$
    uint256 $x$
    uint256[] M;
    uint256[] ID;
uint cnt = 0;
Pseudonym[] PS;
/* Invoked by PSdonyms to verify proof        */
Function verifyTx($pi$, uint256 $x$)
if proof is valid then
    for $i = 0; i <= cnt; i++$ do
        if PS[i].x == $x$ then return false;
    end
    PS[cnt].$address$ = $msg.sender$;
    PS[cnt].$x$ = $x$;
    cnt++;
    return true;
end
return false;
/* Invoked by pseudonyms to add devices       */
Function addDevices($M_i, ID_i$)
for $i = 0;\ i <= cnt;\ i++$ do
    if PS[i]==msg.sender then
        PS[i].M.add($M_i$);
        PS[i].ID.add($ID_i$);
        return true;
    end
end
return false;
/* Invoked by pseudonyms to remove devices */
Function removeDevices($M_i$)
for $i = 0;\ i <= cnt;\ i++$ do
    if PS[i]==msg.sender then
        PS[i].M.delete($M_i$);
        PS[i].ID.delete($ID_i$);
    end
    return true;
end
return false;
/* Invoked by the HSP to query for devices */
Function query($M_i$)
for $i = 0;\ i <= cnt;\ i++$ do
    if PS[i].M contains $M_i$ then return true;
end
return false
```

Pseudonym	Public statement	Token M	ID
PS_i	$\vec{x}_i$	M_1	ID_1
		$\vdots$	$\vdots$
		M_i	ID_i

TABLE 3: Table of authenticated devices in SC

- ID_i sends (ID_i, M_i) to the HSP to start the DHKE. This step corresponds to *(4.2) Key exchange request* in Fig. 2.
- The HSP checks the validity of ID_i by calling the function **query(**M_i**)**. The transaction of **query(**M_i**)** is also verified under the public address of the HSP. If

(M_i) is registered in SC, the HSP accepts M_i as a DHKE key share. This step corresponds to *(4.3) Device query* in Fig. 2.

- Next, the HSP continues the DHKE process by choosing a random $b \in \mathbb{Z}_p$ and computing a token $M_{HSP} = g^b \pmod p$. Afterward, the HSP obtains a signature $\sigma_{HSP} = \mathsf{S\_sign}(M_{HSP}, \mathsf{SK}_{sig})$ and sends (M_{HSP}, σ_{HSP}) to ID_i. This step corresponds to *(4.4) Key exchange response* in Fig. 2.
- ID_i checks if σ_{HSP} is valid.

Step 5: Data sharing

If σ_{HSP} is valid, the HSP and ID_i perform as follows.

- ID_i computes $K_i = {M_{HSP}}^a \pmod p$ and the HSP computes $K_i = {M_i}^b \pmod p$.
- Using symmetric-key ciphers under K_i, ID_i and the HSP securely share data. This step corresponds to *(5) Data sharing* in Fig. 2.
- Finally, if PS_i wants to use ID_i no longer, it can remove ID_i by calling the function **RemoveDevices()** with input M_i.

Fig. 3 describes the flowchart of the proposed system.

V. SECURITY

A. SECURITY REQUIREMENTS

Based on the existing literature [5], [10] in Section II, a secure privacy-preserving health data-sharing system between IoT devices and HSP must meet the following security requirements.

- *User anonymity:* after the initialization phase, the identities of users should be hidden from all other parties while using health services, including the HSP and outside adversaries. It means that it should be difficult for an adversary to gain any information on a user identity from all transactions except the initialization phase.
- *Authentication:* only IoT devices from authorized users can join the system and use healthcare services. This means that it should be difficult for an adversary to illegitimately register either an identity or IoT device to the HSP, even though collusion attacks are allowed.
- *Confidentiality:* health data between IoT devices and the HSP should be transmitted in a secure channel. This means that it should be difficult for an adversary to gain any information on shared health data.

B. SECURITY ANALYSIS

This section analyzes the security of the proposed system and demonstrates that all security requirements above are achieved.

1) Anonymity

in the service initialization phase, U_i sends h_i to the HSP in an insecure channel. To provide user anonymity in subsequent phases, any information on the specific h_i (and the corresponding m_i) should be hidden from an adversary, including the HSP.

Theorem 1. *If the hash function H is collision-resistant, and the zk-SNARK scheme satisfies the zero-knowledge property, the proposed system provides anonymity.*

Proof. We create a series of games from Game_0 to Game_F.

• Game_0 is a real game where the adversary, $\mathcal{A}$, tries breaking the anonymity in real attack scenarios.

• Game_1 is the same as Game_0, except that no collision of H occurs. In this case, assuming that H is collision-resistant, Game_1 is indistinguishable from Game_0. Because the collision-resistance implies onewayness of H, it is also infeasible for $\mathcal{A}$ to find a pre-image m_i of the hash value h_i.

• Game_2 is the same as Game_1, except that the pseudonym with respect to U_i, PS_i is chosen regardless of U_i and h_i. Because PS_i is just a pseudonym generated by a hash output value, it does not link to U_i, meaning that Game_2 is identical to Game_1.

• Game_3 is the same as Game_2, except that π_i presented in the authentication phase (numbered (3) in Fig. 2) between pseudonym PS_i and the SC is simulated, regardless of the relevant m_i (as a witness). In this case, because of the zero-knowledge property of the zk-SNARK scheme, Game_3 is indistinguishable from Game_2. Regarding π_i, the zero-knowledge property of the zk-SNARK shows that π_i does not leak any information on m_i, except for the fact that π_i is associated with one of users $\{U_j\}$ who sent $\{h_j\}$ values to the HSP in the initialization phase. Thus, the zero-knowledge property ensures hiding which h_i (and thus m_i) PS_i uses to prove among the listed $\{h_j\}$ values.

• Game_F is the same as Game_3, except that ID_i is chosen regardless of regardless of U_i and h_i. From our initial assumption that ID_i is not relevant to any user identity from the manufacturing stage, Game_F is identical to Game_3.

Now, we see that Game_F is the simulated game where any information on the target use U_i is not revealed from the viewpoint of $\mathcal{A}$. In addition, in the subsequent phases (numbered (4.1) and (4.2) in Fig. 2), a set of tokens $\{M_i\}$ and IoT devices' identities $\{ID_i\}$ do not disclose any information about their owner U_i, because those values are calculated regardless of user-related values such as m_i and U_i. Instead, they link to the owner's pseudonym PS_i because the pseudonym is used to register devices and tokens to the SC. Hence, using these devices to share data with the HSP does not disclose any information about the user U_i. □

2) Pseudonym Authentication

in the anonymous authentication phase (numbered (3) in Fig. 2), the goal of an adversary is to illegitimately register its pseudonym to the SC, although collusion attacks on witnesses are allowed. These attacks are inevitable (1) when $\mathcal{A}$ registers its pseudonym instead of an honest user, and

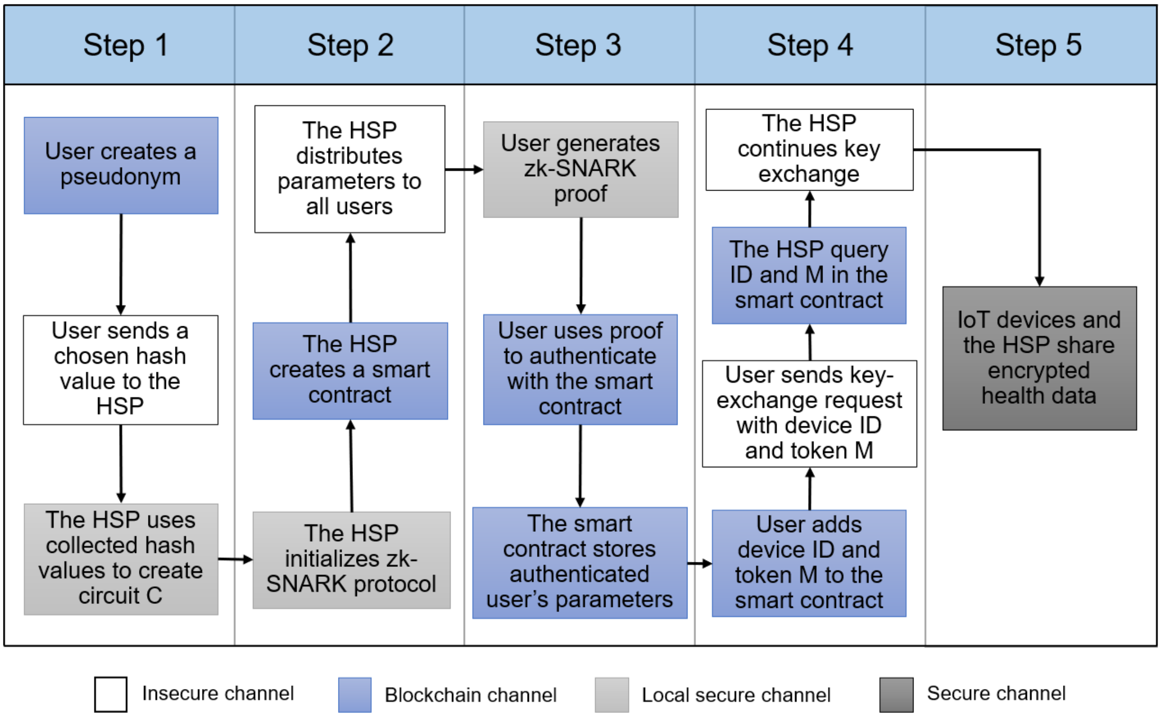

Step 1
Step 2
Step 3
Step 4
Step 5
User creates a pseudonym
User sends a chosen hash value to the HSP
The HSP uses collected hash values to create circuit C
The HSP initializes zk-SNARK protocol
The HSP creates a smart contract
The HSP distributes parameters to all users
User generates zk-SNARK proof
User uses proof to authenticate with the smart contract
The smart contract stores authenticated user's parameters
User adds device ID and token M to the smart contract
User sends key-exchange request with device ID and token M
The HSP query ID and M in the smart contract
The HSP continues key exchange
IoT devices and the HSP share encrypted health data
Insecure channel
Blockchain channel
Local secure channel
Secure channel

FIGURE 3: Flowchart of the proposed system

(2) when $\mathcal{A}$ registers two pseudonyms under the same h_i, knowing the corresponding witness w_i.

Theorem 2. *If the hash function H is collision-resistant, the zk-SNARK scheme satisfies completeness and soundness properties, and the blockchain is immutable, the proposed system provides pseudonym authentication.*

Proof. We create a series of games from Game$_0$ to Game$_F$.

- Game$_0$ is a real game where the adversary, $\mathcal{A}$, tries breaking the pseudonym authentication in real attack scenarios.
- Game$_1$ is the same as Game$_0$, except that no collision of H occurs. Assuming that H is collision-resistant, Game$_1$ is indistinguishable from Game$_0$. As before, the collision-resistance implies that it is infeasible for $\mathcal{A}$ to find a preimage m_i of the hash value h_i.
- Game$_2$ is the same as Game$_1$, except that the SC on the blockchain is unchanged. Because of the immutability property of the blockchain network, Game$_2$ is indistinguishable from Game$_1$. In this case, $\mathcal{A}$ cannot change the verification key on the SC, meaning that the hash values $\{h_i\}$ embedded in the SC cannot be changed by $\mathcal{A}$.
- Game$_3$ is the same as Game$_2$, except that $\mathcal{A}$ succeeds at generating a zk-SNARK proof π without using those $\{h_i\}$ in the SC. Because of the completeness property of zk-SNARK, Game$_3$ is indistinguishable from Game$_2$. In this case, $\mathcal{A}$ cannot generate an accepting proof, based on one or more new and nonregistered $\{h_i\}$ values.
- Game$_4$ is the same as Game$_3$, except that $\mathcal{A}$ succeeds at generating a zk-SNARK proof π without a witness corresponding to the target h_i embedded in the SC. Because of the soundness property of zk-SNARK, Game$_4$ is indistinguishable from Game$_3$. In this case, $\mathcal{A}$ cannot generate an accepting proof, even if $\mathcal{A}$ is given all witnesses but the target m_i.
- Game$_F$ is the same as Game$_4$, except that $\mathcal{A}$ succeeds at registering two pseudonyms under the same h_i, knowing the relevant witness m_i. Because of the soundness property of zk-SNARK and the deterministic public input $\hat{h}_i$, Game$_F$ is indistinguishable from Game$_4$.

To distinguish between Game$_F$ to Game$_4$. $\mathcal{A}$ shares its m_i with others, allowing illegitimate users to use services from the HSP without fair payment. However, as mentioned in Step 3 of Section IV, $\mathcal{A}$ is forced to enter two inputs m_i and $\hat{h}_i$ to circuit C. If $H(m_i)$ is not listed in circuit C or $\hat{h}_i \neq H(m_i + 1)$, the resulting proof is not correctly verified. Under the assumption that H is collision-resistant, $\hat{h}_i$ is uniquely determined by m_i. This means that if m_i is the same, the values of $\hat{h}_i$ must be the same. Thus, $\hat{h}_i$ can be used as a factor for detecting that $\mathcal{A}$ reuses its secret m_i. This is because the zk-SNARK Verify algorithm requires $(\mathsf{VK}, \vec{x}_i, \pi_i)$ as input and $\vec{x}_i$ contains $\hat{h}_i$. If the given $\hat{h}_i$ was used beforehand, the Verify algorithm can reject any proof with the same $\vec{x}_i$. Therefore, the SC will never additionally authenticate malicious users by colluding to share their witnesses $\{m_i\}$. As a result, in Game$_F$, the probability that $\mathcal{A}$ breaks the pseudonym authentication becomes negligible. □

3) IoT device authentication and confidentiality

this phase corresponds to Steps 4 and 5 in our scheme. The goal of an adversary, $\mathcal{A}$, is to illegitimately authenticate its IoT devices to the HSP or try gaining any information on health data transmitted between an IoT device and the HSP. Notably, to protect users' privacy, we assumed that IoT devices do not associate with users' identities.

Theorem 3. *If the underlying digital signature schemes are secure (i.e., unforgeable against chosen message attacks), the blockchain is immutable, DDH problem is difficult to solve, and the underlying SKE scheme is secure, the proposed system provides IoT device authentication and confidentiality (in Steps 4 and 5).*

Proof. We also create a series of games from Game_0 to Game_F, and prove that the probability that $\mathcal{A}$ breaks either the IoT device authentication or confidentiality becomes negligible.

- Game_0 is the real game where $\mathcal{A}$ tries breaking one of those properties in real attack scenarios.
- Game_1 is similar to Game_0, except that $\mathcal{A}$ succeeds at registering an unauthorized device ID under the pseudonym PS_i of an honest user U_i. Unless $\mathcal{A}$ breaks the unforgeability of an underlying signature scheme, it is infeasible for $\mathcal{A}$ to convince the HSP that $\mathcal{A}$ has a legitimate device ID under PS_i. Thus, under the unforgeability of the digital signature scheme, Game_1 is indistinguishable from Game_0.
- Game_2 is the same as Game_1, except that the SC on the blockchain is unchanged. As before, under the immutability property of the blockchain network, Game_2 is indistinguishable from Game_1. In this case, $\mathcal{A}$ cannot change the pairs $\{(ID_i, M_i)\}$ stored on the SC.
- Game_3 is the same as Game_2, except that $\mathcal{A}$ succeeds at modifying a transaction of $\mathbf{query}(M_i)$ issued by the HSP. Under the unforgeability of the digital signature scheme, Game_3 is indistinguishable from Game_2. We see that, in Game_3, $\mathcal{A}$ cannot change two tokens authenticated by a device and the HSP. Thus, even if $\mathcal{A}$ sends (ID_i, M_j) to the HSP as the key exchange request, which is possible because (ID_i, M_j) is transmitted in an insecure channel, such a request will be rejected because of the response to the function $\mathbf{query}(M_j)$. This explains why the proposed system prevents the man-in-the-middle attack against the DHKE protocol.
- Game_4 is the same as Game_3, except that $\mathcal{A}$ succeeds in distinguishing whether a shared key between an IoT device and the HSP is normal or random. Under the DDH assumption, Game_4 is indistinguishable from Game_3.
- Game_F is the same as Game_4, except that $\mathcal{A}$ gains any information on transmitted health data. Under the security of the underlying SKE scheme, Game_F is indistinguishable from Game_4.

In Game_F, it is difficult for $\mathcal{A}$ to launch an eavesdropping attack against transmitted health data. To prevent a stronger version of a chosen-ciphertext attack, we can also consider SKE schemes secure against chosen-ciphertext attacks. As a result, Steps 4 and 5 do not leak any information on a user's identity, bacause (ID_i, M_i) is associated only with their pseudonym PS_i. □

VI. PERFORMANCE EVALUATION

The proposed system is implemented in a device with Intel-Core i7 8700k processor @3700 MHz clock frequency and 32 Gb RAM. The operating system is Windows 10 pro. The blockchain network is the testnet called "Ropsten." The SC is compiled and run in Ethereum Remix on Google Chrome web browser version 95.0.4638.69 (Official Build) (64-bit). The zk-SNARK is initialized using the package, Zokrates [4], version 0.7.10 in WSL 2 Ubuntu 18.04 LTS. The hash function in circuit C is MiMC [25] and the underlying zk-SNARK scheme is Pinocchio [21] based on the elliptic curve of ALT-BN128 [26], [27]. Among several zk-SNARK schemes, such as G16 [28], GM17 [22], and Marlin [29], we chose the Pinocchio scheme because it is nonmalleable and has the lowest computation cost. The MiMC hash function was selected instead of SHA256 because MiMC is designed for zk-SNARK and does not require as much computation power as SHA256.

In our testing system, we use the following five functions of the Zokrates toolbox. The first two functions correspond to the zk-SNARK Setup and KeyGen algorithms, respectively, the second two functions correspond to the zk-SNARK proof algorithm, and the last function corresponds to the zk-SNARK Verify algorithm.

- $compile(C) \rightarrow P$, where C is an arithmetic circuit and P is a set of polynomials.
- $setup(P) \rightarrow (\mathsf{PK}, \mathsf{VK})$, where PK and VK are the proving key and the verification key, respectively.
- $compute - witness(P, m, x) \rightarrow w$, where m is a prover's secret, x is a public statement, and w is a witness.
- $generate - proof(w, \mathsf{PK}) \rightarrow \pi$, where π is a proof.
- $verify(\mathsf{VK}, \pi, x) \rightarrow b$, where b indicates the result of the verification (PASS or FAILED).

Our implementation comprises two parts. The first one is to measure the timings for the zk-SNARK KeyGen, Proof, and Verify algorithms which are performed with different numbers of users. Four circuits are created to accommodate 1,000, 5,000, 10,000, and 20,000 users. The performance evaluation parameters for zk-SNARK are summarized in Table 4. The second one is to calculate the transaction fees for invoking the functions embedded in the SC. With respect to the zk-SNARK verification function, the proof size and the verification cost are almost the same regardless of the different number of users, so our second implementation is tested only for the SC generated for 20,000 users. The performance evaluation parameters for interaction with SC are summarized in Table 6

A. ZK-SNARK PERFORMANCE

To initialize an underlying zk-SNARK scheme, the HSP performs the zk-SNARK Setup and KeyGen algorithms, which can be implemented by running the following command lines. Notably, zk-SNARK parameters are selected randomly by Zokrates (depending on a security parameter).

Parameters	Value
CPU	Core i7 8700k
Clock speed	3700 MHz
RAM	32 Gb
Operating system	WSL 2 Ubuntu 18.04 LTS
zk-SNARK tool	Zokrates v0.7.10
zk-SNARK scheme	PHGR13
Hash function	MiMC
Elliptic curve	ALT-BN128

TABLE 4: Simulation parameters for zk-SNARK scheme

1) *"zokrates compile -i circuit-C.zok -o output"* compiles circuit C from a computer code to the polynomial type. Circuit C is stored as a *.zok* file, named *circuit-C.zok*, and the polynomials are stored in a file named *output*.
2) *"zokrates setup -i output -s pghr13 -b libsnark"* takes polynomials from *output* and generates a pair of PK and VK, using the Pinocchio scheme and backend Libsnark.

To generate proof, the user performs the zk-SNARK Proof algorithm, which is implemented by running the following command lines.

1) *"zokrates compute-witness -a <secret> <public statement> -i output"* computes a witness, using polynomials from *output*, a user's secret and public statement. We assume that the HSP sends *output* to all their valid users as in Step 2 of Section IV.
2) *"zokrates generate-proof -s pghr13 -b libsnark"* uses the witness and PK to generate proof using the Pinocchio scheme and backend Libsnark.

We consider the zk-SNARK Verify algorithm by running the following command line.

1) *"zokrates verify -s pghr13 -b libsnark"* returns "PASS" or "FAILED" indicating a valid or an invalid proof, respectively.

Table 5 presents the performance of the zk-SNARK initialization (including Setup and KeyGen), proof generation, and verification, with respect to computational timings. As the number of users increases, the initialization and proof generation times increase. Moreover the verification time is almost constant regardless of the number of users.

B. INTERACTIONS WITH THE SC

In this subsection, we measure the cost of interacting with the SC. Recall that all interactions with the SC are in the form of transactions, and processing a transaction costs transaction fees. These fees depend on the size of the SC and the complexity of the code (embedded in the SC). Calling a complex function with a large size of parameters needs a higher fee than a simple function with a small size of parameters.

Number of users	Initialization	Proof	Verify
1,000	3.148	2.018	0.219
5,000	12.223	5.319	0.213
10,000	32.164	9.478	0.206
20,000	99.026	18.257	0.181

TABLE 5: zk-SNARK performance (unit: second)

Parameters	Value
Network	Ropsten testnet
IDE	Remix
Browser	Google Chrome v95.0.4638.69
Operating system	Windows 10 pro
Blockchain gateway	Metamask v10.10.2
User's address	0x642fb237b06919393C1B876 6BDEA4Fe57d3cB8eB
HSP's address	0x4D885Cb3a8155bc46f8622d 1A2C132F6B5c17dE2
Smart contract address	0xF8416C862f9D127efBC0fa6 F765775CA14A81755

TABLE 6: Simulation parameters for smart contract

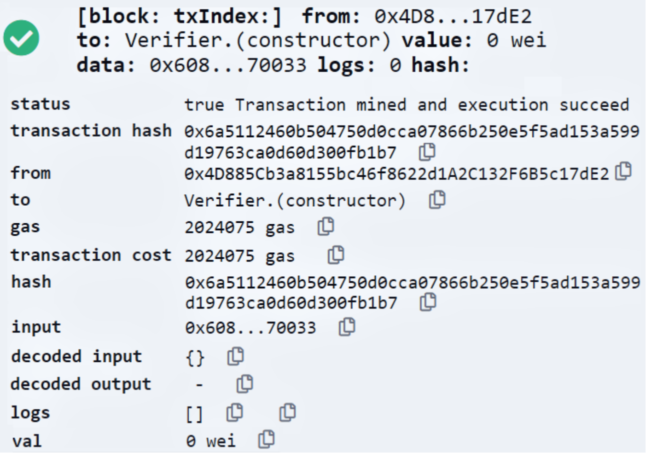
We first create two addresses on the Ethereum network.

- 0x642fb237b06919393C1B8766BDEA4Fe57d3cB8eB for the pseudonym of the user.
- 0x4D885Cb3a8155bc46f8622d1A2C132F6B5c17dE2 for the HSP.

We create an SC with functions described in Algorithm 2. Afterward, it is deployed into the blockchain network with the address 0xF8416C862f9D127efBC0fa6F765775CA14A 81755 As shown in Fig. 4, the cost for deploying this SC is *2,024,075 gas*. The interface of the SC is shown in Fig. 10, containing four functions and a public variable *"qresult"* to show the result of **query**().

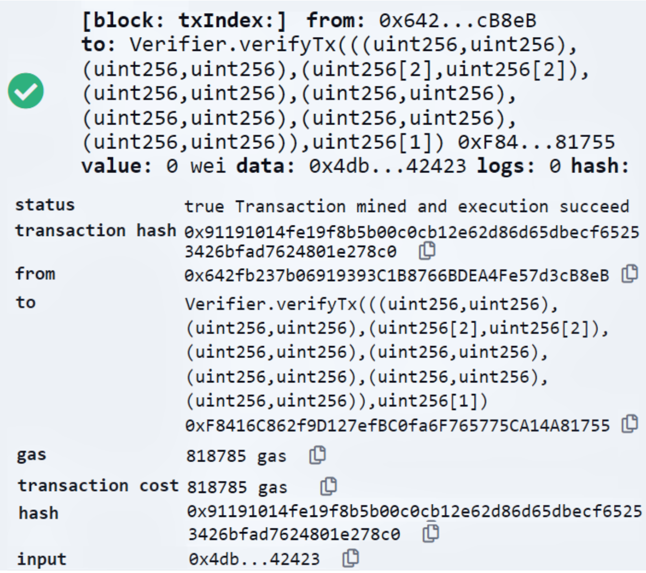
With the SC's address, the pseudonym of the user sends its proof and public statement to the SC to invoke the function **verifyTx**(). The result is shown in Fig. 5. Next, the pseudonym adds its ID_i and M_i to the SC using the function **addDevices**(). The result of this function call is shown in Fig. 6. Afterward, (ID_i, M_i) is sent to the HSP to start the key exchange process. Then, the HSP uses the function **query**() to check the authenticity of M_i as in Fig. 7. The query result is shown in Fig. 9. Finally the pseudonym will remove its device using the function **removeDevices**(). The result is shown in Fig. 8. The transaction fees of running the above functions are shown in Table 7.

In Fig. 5, the field ***from*** indicates the address of the function caller. In this case, it becomes the address of



```
[block: txIndex:]  from: 0x4D8...17dE2
to: Verifier.(constructor) value: 0 wei
data: 0x608...70033 logs: 0 hash:

status            true Transaction mined and execution succeed
transaction hash  0x6a5112460b504750d0cca07866b250e5f5ad153a599
                  d19763ca0d60d300fb1b7
from              0x4D885Cb3a8155bc46f8622d1A2C132F6B5c17dE2
to                Verifier.(constructor)
gas               2024075 gas
transaction cost  2024075 gas
hash              0x6a5112460b504750d0cca07866b250e5f5ad153a599
                  d19763ca0d60d300fb1b7
input             0x608...70033
decoded input     {}
decoded output     -
logs              []
val               0 wei
```

FIGURE 4: Smart contract's deployment



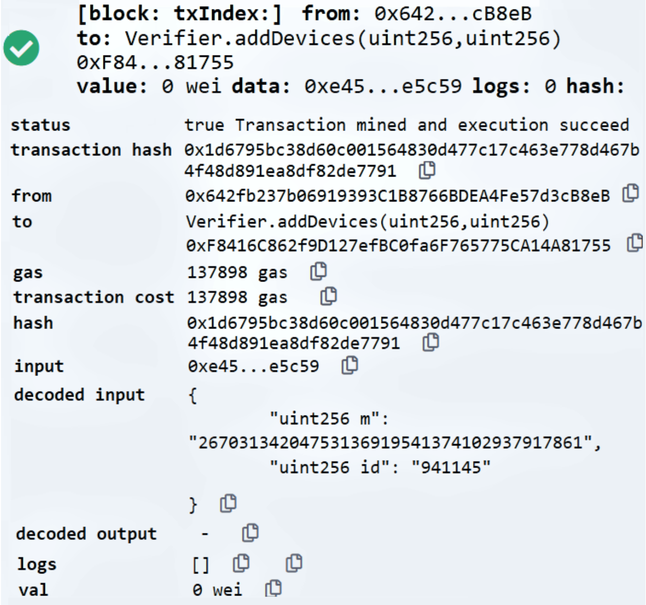
```
[block: txIndex:]  from: 0x642...cB8eB
to: Verifier.verifyTx(((uint256,uint256),
(uint256,uint256),(uint256[2],uint256[2]),
(uint256,uint256),(uint256,uint256),
(uint256,uint256),(uint256,uint256),
(uint256,uint256)),uint256[1]) 0xF84...81755
value: 0 wei data: 0x4db...42423 logs: 0 hash:

status            true Transaction mined and execution succeed
transaction hash  0x91191014fe19f8b5b00c0cb12e62d86d65dbecf6525
                  3426bfad7624801e278c0
from              0x642fb237b06919393C1B8766BDEA4Fe57d3cB8eB
to                Verifier.verifyTx(((uint256,uint256),
                  (uint256,uint256),(uint256[2],uint256[2]),
                  (uint256,uint256),(uint256,uint256),
                  (uint256,uint256),(uint256,uint256),
                  (uint256,uint256)),uint256[1])
                  0xF8416C862f9D127efBC0fa6F765775CA14A81755
gas               818785 gas
transaction cost  818785 gas
hash              0x91191014fe19f8b5b00c0cb12e62d86d65dbecf6525
                  3426bfad7624801e278c0
input             0x4db...42423
```

FIGURE 5: Transaction log of **VerifyTx()**

the pseudonym, 0x642fb237b06919393C1B8766BDEA4Fe57d3cB8eB. The field ***to*** indicates the destination of the function call, which is "Verifier.verifyTx()" of the SC with the address 0xF8416C862f9D127efBC0fa6F765775CA14A81755. The function's inputs are the zk-SNARK proof (comprising elliptic curve points) and the public statement of the pseudonym (i.e., user). This transaction costs 818,785 gas.

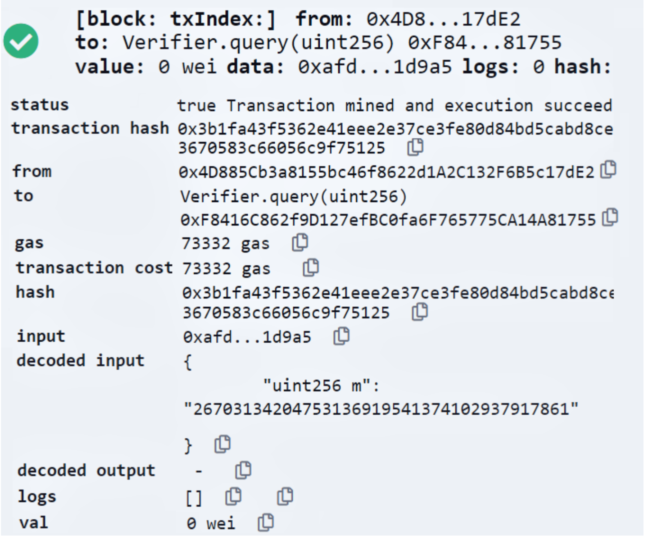
In Fig. 6, the function **addDevices()** takes as inputs two 256-bit unsigned integers. Under the assumption that the DHKE parameters (g, p) are shared in advance by all entities, the first input is calculated as M_i = 26703134204753136919541374102937917861. In addition, the second input is ID_i = 941145 as the identity of an IoT device. In this case, the transaction costs 137,898 gas. These two inputs (M_i, ID_i) will be linked to the pseudonym and stored in the SC as in Table 3.

In Fig. 7, the queried input M_i is in the field **decoded input**. The transaction fee is then 73,332 gas. The result of the function **query** is shown in Fig. 9. It contains two parameters, M_i and ID_i, which are identical to those added by the pseudonym in Fig. 6. It means that (M_i, ID_i) is registered by the legitimate pseudonym. The HSP uses M_i to perform the DHKE process with ID_i.



```
[block: txIndex:]  from: 0x642...cB8eB
to: Verifier.addDevices(uint256,uint256)
0xF84...81755
value: 0 wei data: 0xe45...e5c59 logs: 0 hash:

status            true Transaction mined and execution succeed
transaction hash  0x1d6795bc38d60c001564830d477c17c463e778d467b
                  4f48d891ea8df82de7791
from              0x642fb237b06919393C1B8766BDEA4Fe57d3cB8eB
to                Verifier.addDevices(uint256,uint256)
                  0xF8416C862f9D127efBC0fa6F765775CA14A81755
gas               137898 gas
transaction cost  137898 gas
hash              0x1d6795bc38d60c001564830d477c17c463e778d467b
                  4f48d891ea8df82de7791
input             0xe45...e5c59
decoded input     {
                          "uint256 m":
                  "26703134204753136919541374102937917861",
                          "uint256 id": "941145"

                  }
decoded output     -
logs              []
val               0 wei
```

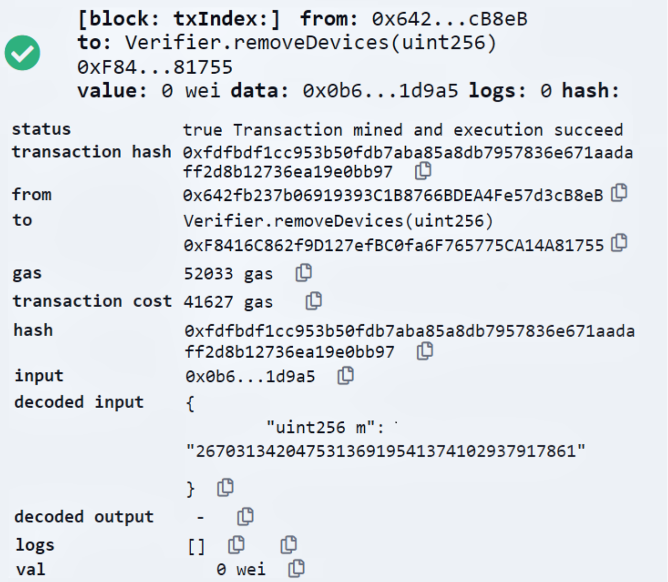
FIGURE 6: Transaction log of **addDevice()**



```
[block: txIndex:]  from: 0x4D8...17dE2
to: Verifier.query(uint256) 0xF84...81755
value: 0 wei data: 0xafd...1d9a5 logs: 0 hash:

status            true Transaction mined and execution succeed
transaction hash  0x3b1fa43f5362e41eee2e37ce3fe80d84bd5cabd8ce
                  3670583c66056c9f75125
from              0x4D885Cb3a8155bc46f8622d1A2C132F6B5c17dE2
to                Verifier.query(uint256)
                  0xF8416C862f9D127efBC0fa6F765775CA14A81755
gas               73332 gas
transaction cost  73332 gas
hash              0x3b1fa43f5362e41eee2e37ce3fe80d84bd5cabd8ce
                  3670583c66056c9f75125
input             0xafd...1d9a5
decoded input     {
                          "uint256 m":
                  "26703134204753136919541374102937917861"

                  }
decoded output     -
logs              []
val               0 wei
```

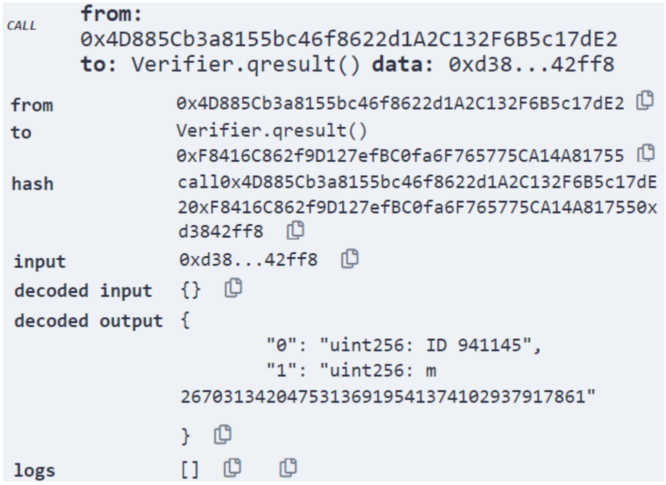
FIGURE 7: Transaction log of **query()**

At this moment, the average transaction fee on the Ethereum network is approximately 5.71 USD. Regarding the transaction fee paid by the HSP, the deployment costs 19.04 USD, but the function **query()** costs only 0.39 USD. Because the deployment is implemented just once and the function **query()** is called many times, the average



```
[block: txIndex:]  from: 0x642...cB8eB
to: Verifier.removeDevices(uint256)
0xF84...81755
value: 0 wei data: 0x0b6...1d9a5 logs: 0 hash:

status            true Transaction mined and execution succeed
transaction hash  0xfdfbdf1cc953b50fdb7aba85a8db7957836e671aada
                  ff2d8b12736ea19e0bb97
from              0x642fb237b06919393C1B8766BDEA4Fe57d3cB8eB
to                Verifier.removeDevices(uint256)
                  0xF8416C862f9D127efBC0fa6F765775CA14A81755
gas               52033 gas
transaction cost  41627 gas
hash              0xfdfbdf1cc953b50fdb7aba85a8db7957836e671aada
                  ff2d8b12736ea19e0bb97
input             0x0b6...1d9a5
decoded input     {
                        "uint256 m":
                  "26703134204753136919541374102937917861"

                  }
decoded output     -
logs              []
val                  0 wei
```

FIGURE 8: Transaction log of **removeDevice()**



```
CALL     from:
         0x4D885Cb3a8155bc46f8622d1A2C132F6B5c17dE2
         to: Verifier.qresult() data: 0xd38...42ff8

from            0x4D885Cb3a8155bc46f8622d1A2C132F6B5c17dE2
to              Verifier.qresult()
                0xF8416C862f9D127efBC0fa6F765775CA14A81755
hash            call0x4D885Cb3a8155bc46f8622d1A2C132F6B5c17dE
                20xF8416C862f9D127efBC0fa6F765775CA14A817550x
                d3842ff8
input           0xd38...42ff8
decoded input   {}
decoded output  {
                      "0": "uint256: ID 941145",
                      "1": "uint256: m
                26703134204753136919541374102937917861"

                }
logs            []
```

FIGURE 9: Query result

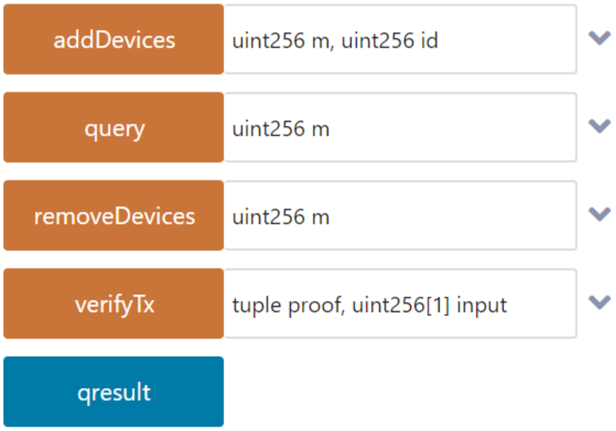


FIGURE 10: Smart contract's interface

Unit	Deployment	Verify Tx	Add devices	Remove devices	Query
Gas	2,024,075	818,785	137,898	73,332	41,627
Ether	0.005062	0.002047	0.000344	0.000183	0.000104
USD	19.04	7.70	1.29	0.69	0.39

TABLE 7: Transaction fees

Data in this table are calculated with a gas price of 0.0000000025 ETH, 1 ETH = 3762 USD.

transaction fee for the HSP can be calculated as follows:

$$\frac{D + wQ}{1 + w},$$

where D is the deployment cost, Q is the cost of **query()**, and w is the number of **query()** queries. In the case when w = 20,000, the average transaction fee for the HSP is 0.391 USD. Regarding the transaction fee paid by the user, the average transaction fee can be calculated as follows:

$$\frac{V + tA + mR}{1 + t + m},$$

where V is the cost of **verifyTx()**, which is called once, A is the cost of **addDevices()**, R is the cost of **removeDevices()**, t is the number of registered devices and m is the number of removed devices. If a user registers for five devices and removes two devices, the average transaction fee for the user is 1.94 USD.

VII. LIMITATIONS AND FUTURE WORK

A. LIMITATIONS

Although the proposed system's privacy is enhanced compared with the existing healthcare systems. it still has limitations.

First, the proposed system uses zk-SNARK as an anonymous authentication technique, incurring a computational burden for users. Despite the verification being fast, the proof generation phase requires a lot of computational power. Hence, it is unsuitable for mobile devices, such as smartphones; users need to use a computer to create their proof instead.

Second, we implemented the proposed system in the Ethereum blockchain network. It results in low performance and high costs in the device management phase. This limitation stems from the core techniques of Ethereum. Ethereum employs the Ethereum Virtual Machine (EVM) blockchain network in which, all nodes must run an EVM separately. Although it makes the network more consistent and fault-tolerant, EVM slows down the network; hence,the number of transactions per second (TPS) is limited. Another reason for the low TPS is the PoW consensus algorithm. Further, the SC language used in Ethereum is Solidity, an object-oriented programming language that depends on the internal state or data on storage and other components. This property results in many unexpected behaviors for function logic in the SC. In other words, the SC can behave differently from what its developer expected or foresaw.

B. FUTURE WORK

Based on the above limitations, future works can follow the direction of decreasing the computational burden of zk-SNARK or inventing a new cryptography technique for anonymous authentication, wherein both proof generation and verification are fast and efficient. Another direction is moving the proposed system to a non-EVM blockchain network, or a network using a PoS algorithm such as Cardano [30] or Polkadot [31]. In Cardano, they use a PoS algorithm called Ouroboros Hydra [32], which supports up to 1,000,000 TPS theoretically. They also use Haskell, a functional programming language, for SCs to reduce unexpected behaviors and make the SC more stable and predictable.

VIII. CONCLUSION

In this study, we proposed a system for anonymous secured health data-sharing between IoT devices and the HSP. The proposed system allows users to hide their identities and encrypt data sent from their IoT devices while using the health services. The zk-SNARK scheme, the DHKE algorithm, and digital signature algorithm are combined to create an anonymous authentication protocol and a secure data sharing channel between IoT devices and the HSP. This scheme also removes the necessity of the third party for countering collusion attacks in the anonymous authentication phase by detecting whether users' witness is used more than once. The blockchain and the SC are used to ensure that anonymous authentication is fair and independent of any single party.

REFERENCES

[1] L. P. Malasinghe, N. Ramzan, and K. P. Dahal, "Remote patient monitoring: a comprehensive study," J. Ambient Intell. Humaniz. Comput., vol. 10, no. 1, pp. 57–76, 2019.

[2] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," J. Cryptol., vol. 1, no. 2, pp. 77–94, 1988.

[3] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in Proceedings of the 23rd USENIX Security Symposium (K. Fu and J. Jung, eds.), pp. 781–796, USENIX Association, 2014.

[4] J. Eberhardt and S. Tai, "Zokrates - scalable privacy-preserving off-chain computations," in IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1084–1091, IEEE, 2018.

[5] A. D. Dwivedi, G. Srivastava, S. Dhar, and R. Singh, "A decentralized privacy-preserving healthcare blockchain for iot," Sensors, vol. 19, no. 2, p. 326, 2019.

[6] S. Badr, I. A. Gomaa, and E. Abd-Elrahman, "Multi-tier blockchain framework for iot-ehrs systems," in The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN) 2018 (E. M. Shakshuki and A. Yasar, eds.), vol. 141 of Procedia Computer Science, pp. 159–166, Elsevier, 2018.

[7] J. Fu, N. Wang, and Y. Cai, "Privacy-preserving in healthcare blockchain systems based on lightweight message sharing," Sensors, vol. 20, no. 7, p. 1898, 2020.

[8] K. N. Griggs, O. Ossipova, C. P. Kohlios, A. N. Baccarini, E. A. Howson, and T. Hayajneh, "Healthcare blockchain system using smart contracts for secure automated remote patient monitoring," J. Medical Syst., vol. 42, no. 7, pp. 130:1–130:7, 2018.

[9] X. Yin, Z. Liu, B. Ndibanje, L. Nkenyereye, and S. M. R. Islam, "An iot-based anonymous function for security and privacy in healthcare sensor networks," Sensors, vol. 19, no. 14, p. 3146, 2019.

[10] R. Attarian and S. Hashemi, "An anonymity communication protocol for security and privacy of clients in iot-based mobile health transactions," Comput. Networks, vol. 190, p. 107976, 2021.

[11] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret: Theory and applications of ring signatures," in Theoretical Computer Science, Essays in Memory of Shimon Even, vol. 3895 of Lecture Notes in Computer Science, pp. 164–186, 2006.

[12] K. M. Hossein, M. E. Esmaeili, T. Dargahi, A. Khonsari, and M. Conti, "Bchealth: A novel blockchain-based privacy-preserving architecture for iot healthcare applications," Comput. Commun., vol. 180, pp. 31–47, 2021.

[13] A. A. Omar, M. Z. A. Bhuiyan, A. Basu, S. Kiyomoto, and M. S. Rahman, "Privacy-friendly platform for healthcare data in cloud based on blockchain environment," Future Gener. Comput. Syst., vol. 95, pp. 511–521, 2019.

[14] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 329–349, ACM, 2019.

[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized Business Review, p. 21260, 2008.

[16] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99) (B. Preneel, ed.), vol. 152 of IFIP Conference Proceedings, pp. 258–272, Kluwer, 1999.

[17] F. Saleh, "Blockchain without Waste: Proof-of-Stake," The Review of Financial Studies, vol. 34, no. 3, pp. 1156–1190, 2020.

[18] V. Buterin, "Ethereum white paper," GitHub repository, vol. 1, pp. 22–23, 2013.

[19] B. Ampel, M. Patton, and H. Chen, "Performance modeling of hyperledger sawtooth blockchain," in 2019 IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 59–61, 2019.

[20] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," CoRR, vol. abs/1801.10228, 2018.

[21] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," Commun. ACM, vol. 59, no. 2, pp. 103–112, 2016.

[22] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Proceedings, Part II (J. Katz and H. Shacham, eds.), vol. 10402 of Lecture Notes in Computer Science, pp. 581–612, Springer, 2017.

[23] J. M. Miret, D. Sadornil, and J. Tena, "Pairing-based cryptography on elliptic curves," Math. Comput. Sci., vol. 12, no. 3, pp. 309–318, 2018.

[24] J. F. Barrera, C. Vargas, M. Tebaldi, and R. Torroba, "Chosen-plaintext attack on a joint transform correlator encrypting system," Optics Communications, vol. 283, no. 20, pp. 3917–3921, 2010.

[25] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part I (J. H. Cheon and T. Takagi, eds.), vol. 10031 of Lecture Notes in Computer Science, pp. 191–219, 2016.

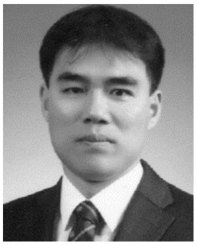
[26] C. Reitwiessner, "Eip-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128," Ethereum Improvement Proposals, vol. no. 196, February 2017.

[27] V. Buterin and C. Reitwiessner, "Eip-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128," Ethereum Improvement Proposals, vol. no. 197, February 2017.

[28] J. Groth, "On the size of pairing-based non-interactive arguments," in Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II (M. Fischlin and J. Coron, eds.), vol. 9666 of Lecture Notes in Computer Science, pp. 305–326, Springer, 2016.

[29] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward, "Marlin: Preprocessing zksnarks with universal and updatable SRS," IACR Cryptol. ePrint Arch., p. 1047, 2019.
[30] C. Hoskinson, "Why we are building cardano," White paper., 2017.
[31] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," White Paper, vol. 21, pp. 2327–4662, 2016.
[32] M. M. T. Chakravarty, S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias, and A. Russell, "Hydra: Fast isomorphic state channels." Cryptology ePrint Archive, Report 2020/299, 2020.

DUC ANH LUONG received a bachelor of computer science from Sangmyung University. His research interests include blockchain technology, IoT, and information security.



JONG HWAN PARK received the B.S. degree from the Department of Mathematics, Korea University, Seoul, South Korea, in 1999, and the M.S. and Ph.D. degrees with the Graduate School of Information Security, Korea University, Seoul, South Korea, in 2004 and 2008, respectively.

From 2009 to 2011, he was a Research Professor with Kyung Hee University, and from 2011 to 2013 he was a Research Professor with Korea University. Since 2013, he has been an Assistant Professor with the Department of Computer Science, Sangmyung University, Seoul, South Korea. His research interests include functional encryption, broadcast encryption, authenticated encryption, and various cryptographic protocols.