Received November 17, 2021, accepted November 22, 2021, date of publication November 25, 2021, date of current version December 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3130552

MEXchange: A Privacy-Preserving Blockchain-Based Framework for Health Information Exchange Using Ring Signature and Stealth Address

DEOKSANG LEE[1] **AND MINSEOK SONG**[1,2], **(Member, IEEE)**
[1]Department of Industrial & Management Engineering, Pohang University of Science and Technology (POSTECH), Pohang 37673, Republic of Korea
[2]Open Innovation Bigdata Center, Pohang University of Science and Technology (POSTECH), Pohang 37673, Republic of Korea

Corresponding author: Minseok Song (mssong@postech.ac.kr)

This work was supported by the Ministry of Science and ICT (MSIT), South Korea, under the Information Technology Research Center (ITRC) Support Program through the Institute for Information & Communications Technology Planning & Evaluation (IITP) under Grant IITP-2018-0-01441.

ABSTRACT Health information exchange (HIE) refers to the integrated management and secure sharing of health information among healthcare entities. HIE improves healthcare quality and streamlines healthcare administrative work. These advantages have propelled health- care stakeholders to implement HIE. However, challenged by issues such as security, privacy, and costs, HIE is not widespread. Recent studies have suggested blockchain-based HIE for solving security and privacy issues. Unfortunately, existing blockchain-based HIE studies do not consider the privacy issues caused by analyzing senders and receivers of transactions in the blockchain. In this work, we suggest MEXchange, a novel blockchain-based privacy-preserving HIE that prevents the privacy issue by obscuring the sender and concealing receiver addresses. We propose smart contracts and workflow that use ring signature and stealth address for blockchain-based HIE. Software components and implementation of MEXchange on the Ethereum private network are discussed. We evaluate MEXchange quantitatively by measuring the transaction latency and throughput of exchanging. Also, we evaluate MEXchange qualitatively using the requirements of the Office of National Coordinator for Health Information Technology (ONC). Moreover, we proceed with threat modeling based on STRIDE. Finally, we compare MEXchange with Ancile, FHIRChain, Integrating the Healthcare Enterprise Cross-Enterprise Document Sharing (IHE XDS), and MedRec. The MEXchange lowers barriers to the application of blockchain-based HIE systems by mitigating privacy and security issues among healthcare stakeholders.

INDEX TERMS Health information exchange, blockchain, ring signature, stealth address, privacy.

I. INTRODUCTION

Health information exchange (HIE) defines the integrated management and secure sharing of health information among healthcare entities [1]. The HIE improves the quality of health care, streamlines hospital care and administrative work, and enables diverse data analysis of research institutes [2]. These advantages have facilitated several research and developments geared to implement HIE. However, security and privacy issues impede this realization [3], [4].

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Imran Tariq.

Healthcare stakeholders are attempting to address security and privacy issues in HIE through blockchain adoption [5]–[16]. To mention a few studies, Azaria *et al.* [5] sought to enhance interoperability by applying blockchain to health information exchange. Badr *et al.* [15] presented a secure healthcare data sharing method on blockchain considering both health information and IoT measured data. Patel *et al.* [16] utilized blockchain and cryptographic techniques to present a framework suitable and safe for medical image sharing.

The benefits of blockchain in HIE come in three aspects: integrity, availability, and trust. To falsify data in the blockchain, attackers alter all the data within the first and the latest block; thus, it is impossible to manipulate data in the blockchain [8]. Since blockchain reduces the number of centralized servers needed to implement HIE, new attack surfaces shrink [5]. Blockchain was invented assuming a trustless environment, verifying the trust of participants is not necessary, which increases the robustness of the overall system [6]. Several countries are examining the possibility of blockchain-based HIE. For instance, the U.S. announced that it will adopt blockchain to the management of electronic medical records (EMRs). China considers blockchain technology to strengthen security and facilitate HIE [17].

However, prior studies allow attackers to infer treatment patterns and hospital visit history by analyzing senders and receivers of transactions, resulting in de-anonymizing patients' identities. In finance, some studies identify participants in Bitcoin [18], [19]. For example, Flender *et al.* derived a Bitcoin transaction graph and matched it with data obtained from web scraping. Consequently, they identified some participants in the Bitcoin network [18]. Prior blockchain-based HIE research also pointed out this issue. MedRec states that "even without the direct disclosure of a patient name, inference about who a particular patient is could be drawn from metadata of one Ethereum address with multiple others" [20]. Also, Ancile mentioned that blockchain analysis can reveal the visit frequency of a patient to a specific hospital and sensitive information such as date of birth, medical condition, and residential area [7].

This paper aims to propose and implement a novel blockchain-based privacy-preserving framework, MEXchange, as a solution to the inference problem through obscuring and concealing the sender and receiver addresses, respectively. Specifically, we adopt the ring signature and stealth address. Ring signature employs fake senders to obscure a genuine sender [21] while stealth address specifies a one-time address as the recipient and provides clues that allow the genuine receiver to confirm what was delivered to him [22]. We propose a system architecture, smart contracts, workflow, and software components for MEXchange. The system architecture shows the high-level view of MEXchange, which includes the role of players and interactions between the players and smart contracts. Also, we present generalized smart contracts to enable MEXchange implementation on any blockchain platforms that support smart contracts. The workflow outlines exchange steps and covers the detailed interaction with smart contracts in each step. The software components are suggested for MEXchange prototype implementation. Lastly, we implement MEXchange on Ethereum and evaluate it in quantitative and qualitative aspects. In the quantitative aspect, we measure the transaction latency and throughput. In the qualitative aspect, we evaluate MEXchange in eight dimensions: authenticity, availability, confidentiality, ease of integrating new data providers, integrity, privacy, scalability, and transparency. Finally, we identify the threats in MEXchange through threat modeling based on STRIDE.

The remainder of the paper is organized as follows. Section II reviews related articles on blockchain-based HIE; Section III introduces the prior knowledge required to understand MEXchange; Section IV presents a system architecture, smart contracts, workflow, and software components. Furthermore, results of implementation and evaluation are described in Section V; Section VI compares MEXchange with other HIE systems; Finally, Section VII concludes this work.

II. RELATED WORKS

Recently, studies have combined blockchain with HIE to enhance security and privacy. Azaria *et al.* [5] proposed the first blockchain-based HIE approach, MedRec. Suppose that EMRs are stored in hospitals' private databases, MedRec suggested an incentive and access control mechanism for exchanging EMRs on using Ethereum. Furthermore, it designed smart contracts with interactions between healthcare entities and blockchain. Although pioneer to blockchain-based HIE research, MedRec does not include the measures to prevent security and privacy issues to arise with blockchain in HIE.

Several studies applied cryptographic techniques to blockchain-based HIE for improving security. FHIRChain [6] proposed a token-based access control mechanism that encrypts metadata with recipients' public keys to allow recipients to decrypt the metadata and obtain EMRs. The metadata is data needed to achieve health information, such as IP and PORT. FHIRChain analyzed the security requirements of the Office of National Coordinator for Health Information Technology (ONC) and reflected them in the system design. Liu *et al.* [23] proposed a secure access control mechanism BPDS to share EMRs using ciphertext-policy attribute-based encryption (CP-ABE). The CP-ABE employs attributes such as ID and name to encrypt and decrypt data. BPDS allows participants who hold correct attributes can decrypt metadata and access EMRs, which results in preventing unwanted data disclosure. Ancile [7] improves security through a proxy re-encryption scheme that streamlines key sharing steps. This leads to reducing the possibility of key theft by decreasing the number of key shares. Moreover, Ancile designed smart contracts and workflow suitable for sharing EMRs with proxy re-encryption.

Alternatively, some researchers have strived to enhance security and privacy by limiting blockchain participants. Omar *et al.* [24] proposed MediBchain that employs a permissioned blockchain, which permits authorized users to participate in the blockchain. MediBchain uses private accessible units as intermediaries for storing health information in the blockchain. Similarly, data recipients receive health information via private access units and so, data disclosure does not happen unless the private accessible units are compromised. Zhang *et al.* [25] proposed the blockchain-based HIE that exploits private and consortium blockchain where an individual or organizations manage the blockchain participants. Private blockchain stores patients' health information securely, while consortium blockchain manages metadata for sharing.

Although Ancile and FHIRChain proposed the blockchain-based HIE approaches with cryptographic methods, the inference problem was not considered. Although they increase confidentiality by encrypting sensitive data. the inference problem remains because the sender and receiver are still publicly visible. Omar *et al.* and Zhang *et al.* limited participants to authorized users. However, attackers can de-anonymize participants using senders and receivers in transactions when the authorized nodes are compromised. Thus, it is necessary to find a method that prevents the inference problem.

III. PRELIMINARIES

This section provides the necessary background information to understand MEXchange. First, we explain the smart contracts used for implementing business logic in the blockchain. Second, we describe the generation and verification of the ring signature to obscure the sender of a transaction. Finally, we explain how to create and verify the stealth address to hide the recipient.

A. SMART CONTRACTS

Smart contracts are programs for managing digital assets in a decentralized manner according to the business logic specified in the contracts [26]. They are deployed and executed through transactions [27]. Business owners who want to deploy smart contracts generate and broadcast transactions containing smart contracts to the blockchain network. The smart contracts deployment is complete when a block containing the transactions is attached to the blockchain. When executing smart contracts, business stakeholders input parameters in transactions and transfer them to the blockchain network. Then, blockchain participants execute smart contracts using the parameters and update variables. Upon deploying smart contracts, manipulating their contents is hardly possible. In this paper, we designed six smart contracts to share health information securely and prevent data falsification.

B. RING SIGNATURE

Ring signature employs fake senders not participating in a transaction to obscure a genuine sender while providing the receiver with clues used to verify the transaction. [21]. With the clues, the receiver can notice that a particular sender holds a private key corresponding to one of the public keys in the signature. Since most blockchain platforms adopt Elliptic Curve Cryptography (ECC), this paper employs an ECC-based ring signature method [28].

Suppose a sender s sends a ring-signed transaction with ring size n to a receiver, and other participants' public keys B are known in advance. The sender randomly selects the initial value k and generates an initial point T_i as below:

$$T_i = (x_i, y_i) = k \cdot G \tag{1}$$

where x_i and y_i are coordinates of the point T_i and $\cdot$ denotes the elliptic curve multiplication. The sender randomly selects u_t where $t = i+1, i+2, \ldots, n, 1, 2, \ldots, i-1$. Subsequently, the sender calculates c_t and T_t ($t = i+1, i+2, \ldots, n, 1, 2, \ldots, i-1$ and $t-1 = n$ when $t = 1$) as below:

$$\begin{aligned} c_t &= hash(x_{t-1}) &(2)\\ T_t &= (x_t, y_t) = u_t \cdot G + c_t \cdot B_t &(3)\end{aligned}$$

where *hash* is the hash function. Then, c_i and u_i are derived as below:

$$\begin{aligned} c_i &= hash(x_{i-1}) &(4)\\ u_i &= v - d_i c_i \ mod\ q &(5)\end{aligned}$$

where d_i is the private key of the sender and q is the order of the elliptic curve. The sender generates a transaction containing ring signature $\sigma = (c_1, u_1, \ldots, u_n, B_1, \ldots, B_n)$ and sends it to the receiver.

After receiving the transaction, the receiver verifies it using σ. First, the receiver calculates T_t and c_{t+1} for all t $(1 \le t \le n-1)$ using equations (2) and (3). Second, the receiver derives T_n and c'_1 as below:

$$\begin{aligned} T_n &= (x_n, y_n) = u_n \cdot G + c_n \cdot B_n &(6)\\ c'_1 &= hash(x_n) &(7)\end{aligned}$$

Finally, the receiver checks whether the c'_1 equals c_1 or not. If the values are the same, the ring signature verification is completed.

C. STEALTH ADDRESS

Stealth address uses a one-time address to improve privacy by disconnecting a link between the sender and receiver of a transaction in a way that does not specify the recipient's address [22]. Although the sender's address is absent, the transaction contains clues that the recipient can figure out. The receiver finds transactions sent to him with the clues and private key. In the following, we explain the details of creating and verifying the stealth address.

When sending a transaction to a receiver with stealth address, the sender selects a private key k randomly and calculates the corresponding public key K. The public key K is computed by:

$$K = k \cdot G \tag{8}$$

Using the public key B of the receiver, the sender derives a stealth address SA_{tx} calculated by:

$$SA_{tx} = hash(hash(k \cdot B) \cdot G + B) \tag{9}$$

Then, the sender creates a transaction including K and SA_{tx} and sends it to the blockchain network.

The receiver derives the stealth private, public key, and stealth address using the receiver's private key b. The stealth private key *sPriv* is driven by:

$$sPriv = hash(b \cdot K) + b \tag{10}$$

Subsequently, the receiver calculate stealth public key *sPub* as below:

$$sPub = sPriv \cdot G \tag{11}$$

Finally, the receiver calculates stealth address SA'_{tx} using stealth public key:

$$SA'_{tx} = hash(sPub) \tag{12}$$

If SA'_{tx} equals to SA_{tx}, the receiver concludes that the transactions is transferred to him.

IV. METHODS

This section describes methods to design MEXchange, a system that utilizes ring signature and stealth address with blockchain-based HIE. A general over-view is presented, and then we explain smart contracts, workflow, and software components for MEXchange. Furthermore, we demonstrate how we implement the prototype of MEXchange and evaluate it.

A. MEXchange OVERVIEW

Figure 1 illustrates the MEXchange overview we devised to address the inference problem. In MEXchange, there are four players: certificate authority, hospitals, patients, and requestors. Certificate Authority authenticates new participants and records them in the blockchain. The hospitals manage health information in their databases and provide requestors health information. Patients grant access to the requestors and provide them the metadata received from the hospitals for exchanging health information. Requestors ask patients for access and request health information from the hospitals.

We employ a private blockchain so that only authorized users can participate in MEXchange. Furthermore, we assume that the transaction fee is zero since we do not utilize public blockchain. To leverage the blockchain, we employ six smart contracts: access permit, access request, exchange log, information hash, participant, and ring verifier contracts. Access permit and request contracts govern access permission granted by patients and access requests from requestors respectively. Exchange log contract stores the history of exchange between the data providers and requestors for audit. Information hash contract stores the hash values of health information and metadata for data integrity. Also, it manages the encrypted metadata, stealth address, and ephemeral key to allow patients to recover their metadata. Participant contract manages the participants' identity data such as blockchain addresses and public keys. Finally, the ring verifier contract verifies the ring signature included in the access request.

We utilize ring signature and stealth address to enhance privacy. When asking for access permission from a patient, a requestor generates a transaction with fake senders and a receiver. The patient scans stealth addresses in transactions using the clues and finds the access request forwarded to him. Thus, the sender and receiver are not exposed to blockchain participants and this prevents the inference problem.

B. SMART CONTRACTS

We defined six smart contracts in aspects of variables and functions as shown in Figure 2. Each smart contract stores data in variables and manage them through functions. These six smart contracts consist of access permit, access request, exchange log, information hash, participant, and ring verifier contracts.

1) ACCESS PERMIT CONTRACT

Access Permit Contracts (APC) manages the access permission by patients. The APC contains five variables:

- $addrs_{ARC}$: the addresses of access request contracts
- *perID*: the access permission ID
- $enc[accPerForm]_{sym}$: the access permission form encrypted with a symmetric key. The access permission form contains the patient's address, the patient's signature, the requestor's address, health information ID, the address of the Information Hash Contract(IHC), and metadata, i.e., $\langle addr_{pat}, sig_{pat}, addr_{req}, ID_{HI}, addr_{IHC}, metadata\rangle$. The sig_{pat} is used by the requestor and hospital to check whether the permission is from the intended patient. The patient generates the sig_{pat} by signing a simple message consented on the MEXChange such as 'yes.' The requestor and hospital verify it using the patient's public key and the message.
- $enc[sym]_{req}$: the symmetric key encrypted with the public key of the requestor
- $enc[sym]_{hos}$: the symmetric key encrypted with the hospital's public key

Upon receiving an access permission by the patient, the function *storeAccPer* generates new *reqID* and stores *reqID*, $enc[accPerForm]_{sym}$, $enc[sym]_{req}$, and $enc[sym]_{pat}$ in variables. Additionally, the *storeAccPer* is only called by the Access Request Contracts.

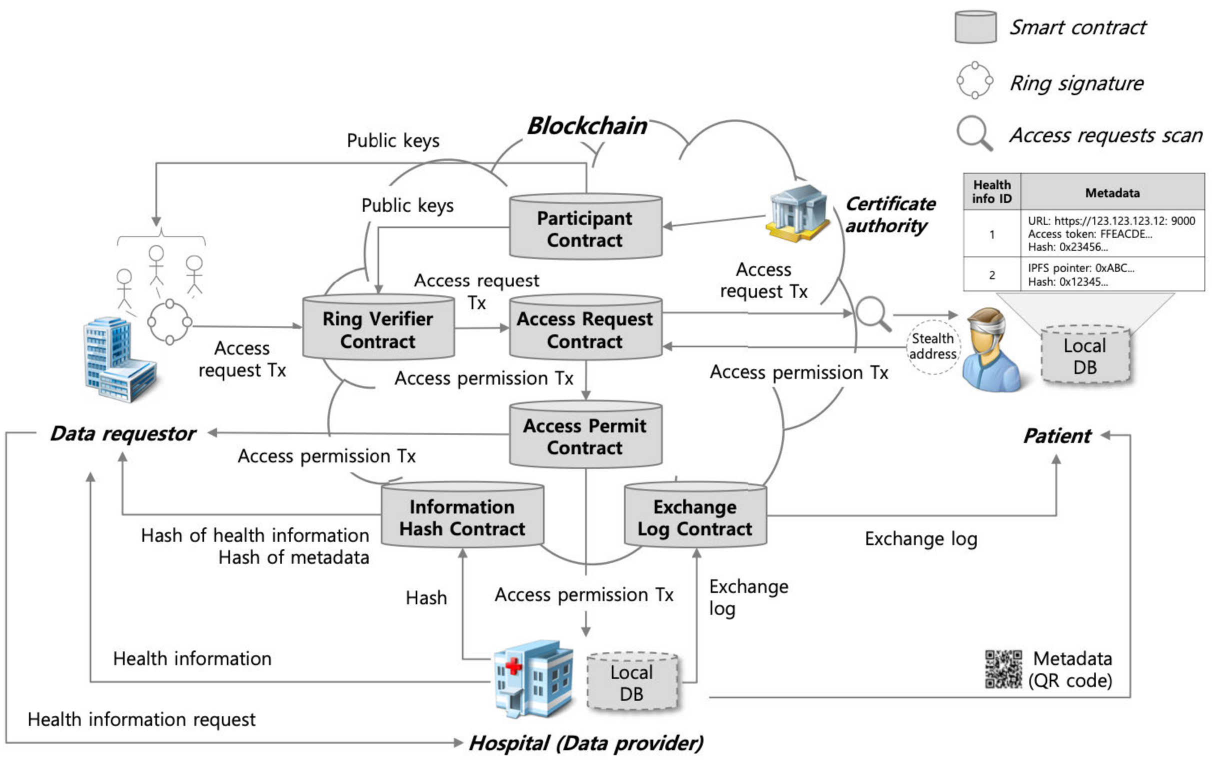
Algorithm 1 storeAccPer

```
 1: Input: $enc[accPerForm]_{sym}$, $enc[sym]_{req}$
 2: $enc[sym]_{hos}$
 3: Output: perID
 4: if The address of sender is in $addrs_{ARC}$ then
 5:     $perID \leftarrow perID + 1$
 6:     Store $\langle perID, enc[accPerForm]_{sym}, enc[sym]_{req},$
 7:     $enc[sym]_{hos}\rangle$
 8:     Return perID
 9: else
10:     Return void
11: end if
```

2) ACCESS REQUEST CONTRACT

Access Request Contract (ARC) manages access requests forwarded to patients. The ARC holds eight variables:

- $addr_{RVC}$: the address of the Ring Verifier Contract
- $addr_{APC}$: the address of the Access Permit Contract
- *reqID*: the access request ID
- *SA*: stealth address generated using the patient's public key
- *ephemKey*: the clue used to calculate stealth address



Health info ID	Metadata
1	URL: https://123.123.123.12: 9000 Access token: FFEACDE... Hash: 0x23456...
2	IPFS pointer: 0xABC... Hash: 0x12345...

FIGURE 1. MEXchange overview.

- $enc[accReqForm]_{sym}$: access request form encrypted with a symmetric key. The access request form consists of the requestor's signature, the addresses of the requestor, and the ID of health information, i.e., $\langle sig_{req}, addr_{req}, ID_{HI} \rangle$
- $enc[sym]_{pat}$: the symmetric key encrypted with the patient's public key
- *state*: the state of access request such as 'None,' 'permitted,' and 'rejected'
- *perID*: the ID of corresponding access permission in the APC

The function *storeAccReq* stores $\langle reqID, SA, ephemKey, enc[sym]_{pat}, enc[accReqForm]_{sym}$ in variables. Also, this function is only called by Ring Verifier Contract(RVC).

3) EXCHANGE LOG CONTRACT

Exchange Log Contract (ELC) preserves logs (e.g., requestor's ID and timestamp) so that patients and hospitals can monitor exchanging. The ELC includes five variables:

- $addr_{hos}$: hospital's address that owns the ELC
- ID_{HI}: health information ID generated by the hospital's health information system
- $enc[log]_{sym}$: the log encrypted with a symmetric key
- $enc[sym]_{pat}$: the symmetric key encrypted with the patient's public key

Algorithm 2 storeAccReq

```
1:  Input: $SA$, $EphemKey$, $enc[accReqForm]_{sym}$
2:  $enc[sym]_{pat}$
3:  Output: $reqID$
4:  if The address of sender equals to ring verifier contract
    then
5:      $reqID \leftarrow reqID + 1$
6:      $state \leftarrow$ 'None'
7:      $perID \leftarrow None$
8:      Store $\langle reqID, SA, ephemKey, enc[accReqForm]_{sym},$
9:      $enc[sym]_{pat}, state, perID \rangle$
10:     Return $reqID$
11: else
12:     Return $-1$
13: end if
```

- $enc[sym]_{hos}$: the symmetric key encrypted with the hospital's public key

To be more specific about the ID_{HI}, the health information ID is provided to the patients after the patients are treated and health information is generated. Then, the patients can acquire exchange log for their health information using the ID later. The function *storeLog* allows the hospital to store the logs in the ELC. The hospital whose address matches with $addr_{hos}$ can store the log in the ELC.

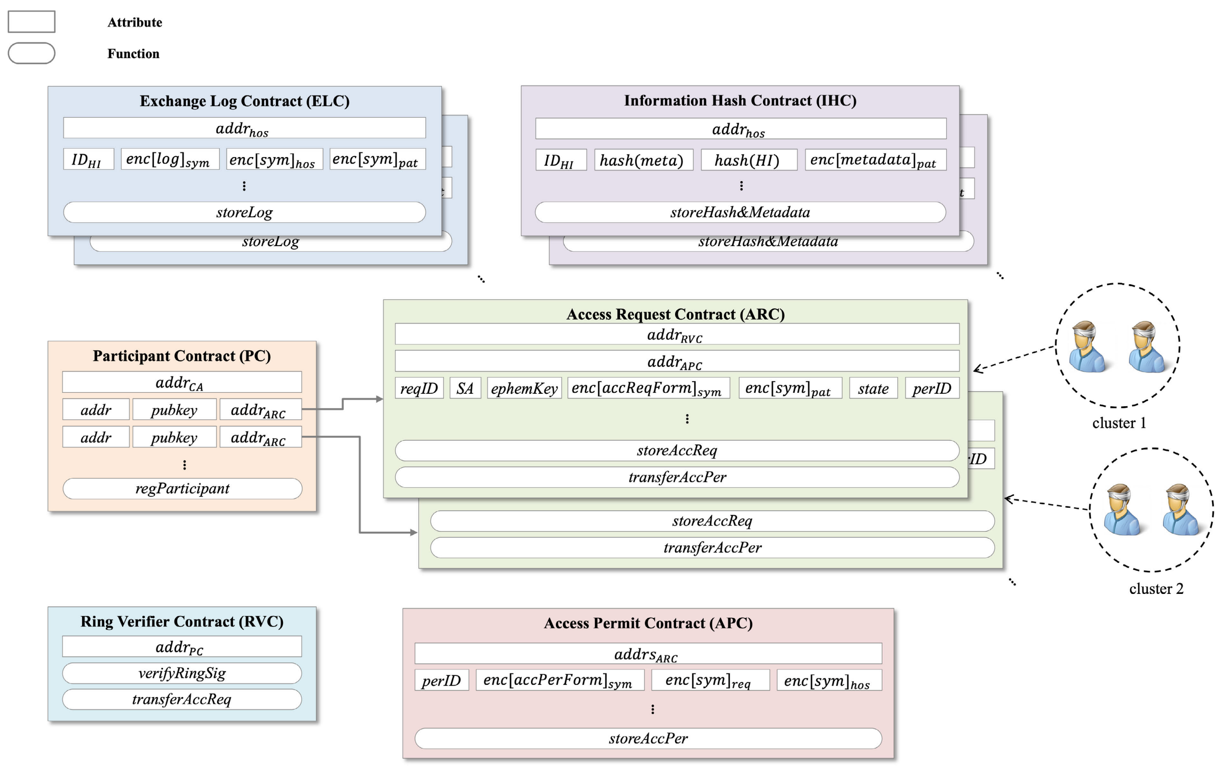


FIGURE 2. Smart contracts in MEXchange.

Algorithm 3 transferAccPer
```
1: Input: $reqID$, $enc[accPerForm]_{sym}$, $enc[sym]_{req}$
2: $enc[sym]_{hos}$
3: Output: void
4: Obtain $SA$ corresponding to $reqID$
5: if The address of sender equals to $SA$ then
6:     $result \leftarrow$ APC.$storeAccPer(enc[accPerForm]_{sym}$,
7:     $enc[sym]_{hos}, enc[sym]_{pat})$
8:     $perID \leftarrow result$
9:     $state \leftarrow$ 'permitted'
10: else
11:     Discard the access permission
12: end if
```

4) INFORMATION HASH CONTRACT

Information Hash Contract (IHC) is responsible for assuring the integrity of health information. To achieve integrity, the IHC holds four variables:

- $addr_{hos}$: the address of a hospital that owns the IHC
- ID_{HI}: the ID of health information
- $hash(meta)$: the hash of metadata used to verify the integrity of metadata
- $hash(HI)$: the hash of health information used to verify the integrity of health information

The function *storeHash* allows hospitals to store the set of variables ID_{HI}, $hash(meta)$, and $hash(HI)$ in this contract. This function is only invoked by the hospital whose address is $addr_{hos}$.

Algorithm 4 storeLog
```
1: Input: $addr_{hos}$, $ID_{HI}$, $enc[log]_{sym}$, $enc[sym]_{hos}$
2: $enc[sym]_{pat}$
3: Output: Whether the log is stored or not
4: if The address of sender equals to $addr_{hos}$ then
5:     Store $\langle ID_{HI}, enc[log]_{sym}, enc[sym]_{hos}, enc[sym]_{pat} \rangle$
6:     Return True
7: else
8:     Return False
9: end if
```

Algorithm 5 storeHash
```
1: Input: $addr_{hos}$, $ID_{HI}$, $hash(meta)$, $hash(HI)$
2: Output: void
3: if The address of sender equals to $addr_{hos}$ then
4:     Store $\langle ID_{HI}, hash(meta), hash(HI) \rangle$
5: else
6:     Discard the transaction
7: end if
```

5) PARTICIPANT CONTRACT

Participant Contract (PC) owned by the CA governs the participants' identity data to manage participants. The PC contains five variables as below:

- $addr_{CA}$: the address of CA
- $addr$: the participant's address
- $pubkey$: the participant's public key

- *addr*: the participant's address
- $addr_{ARC}$: the address of ARC to which the patient belongs, which is used by requestors for requesting access

The PC provides the function *regParticipant* to allow the CA to store the participants' data.

Algorithm 6 regParticipant

```
1: Input: $addr_{CA}$, $address$, $pubkey$, $addr_{ARC}$
2: Output: whether the registration is success or not
3: if The address of sender equals to $addr_{CA}$ then
4:     Store $\langle address, pubkey, addr_{ARC} \rangle$
5:     Return True
6: else
7:     Return False
8: end if
```

6) RING VERIFIER CONTRACT

Ring Verifier Contract (RVC), governed by CA, verifies the ring signature in the access request transactions. The RVC contains the address of the PC $addr_{PC}$ as a variable. The PC manages all participants who registered in MEXchange. Also, the PC is the only smart contract deployed in the blockchain. Therefore, even if the RVC includes $addr_{PC}$, the specific participants cannot be linked. The function *verifyRingSig* is responsible for verifying the ring signature. After verification is complete, the RVC transfers the access requests to the ARC.

Algorithm 7 verifyRingSig

```
1: Input: $sig_{ring} = (c_1, u_1, u_2, \ldots, u_n, addr_1, \ldots, addr_n)$
2: Output: Whether the verification succeeds or not
3: $B \leftarrow$ Obtain the public keys corresponding to addresses
   from the PC
4: for Iteration $i = 1, 2, \ldots, n$ do
5:     if i = 1 then
6:         $(x_i, y_i) \leftarrow u_i \cdot G + c_1 \cdot B_i$
7:     else
8:         $(x_i, y_i) \leftarrow u_i \cdot G + x_i \cdot B_i$
9:     end if
10: end for
11: $c'_1 \leftarrow hash(x_n)$
12: if $c'_1 = c_1$ then
13:     Return True
14: else
15:     Return False
16: end if
```

C. WORKFLOW

This section describes five workflows: participants registration, metadata preparation, access request, access permission, and health information sharing.

Algorithm 8 transferAccReq

```
1: Input: $sig_{ring}$, $ARCaddr$, $SA$, $ephemKey$, $enc[sym]_{pat}$,
   $enc[accReqForm]_{sym}$
2: if verifyRingSig($sig_{ring}$) = True then
3:     Invoke the $storeAccReq$ of the ARC whose address
       equals to $ARCaddr$
4: end if
```

1) PARTICIPANT REGISTRATION WORKFLOW

Participant registration workflow authenticates users and registers them as participants of MEXchange. The procedure of this workflow is described below:

- A user generates private key, public key, and address, i.e., *privkey*, *pubkey*, and *addr*
- The user sends *pubkey* and *addr* to the CA with signature *sig* signed using the private key
- The CA verifies the *sig* using the *pubkey*
- The CA selects $addr_{ARC}$ to which the user will belong and store $\langle pubkey, addr, addr_{ARC} \rangle$ in the PC using the function *regParticipant*
- After the registration is complete, the CA delivers *ID* and $addr_{ARC}$ to the user

2) METADATA PREPARATION WORKFLOW

In metadata preparation workflow, a hospital generates and provides metadata to a patient. Then, the patient receives the metadata and store it in the local database. The metadata is data needed to achieve health information such as IP, PORT, and database query. The detailed workflow proceeds as follows:

- The hospital stores health information of treated patients in the local database
- The hospital generates metadata and calculates hash values $hash(meta)$, $hash(HI)$
- The hospital stores $\langle ID_{HI}, hash(meta), hash(HI) \rangle$ in the IHC using *storeHash*
- The hospital provides the ID_{HI} and metadata for the patient
- The patient stores $\langle ID_{HI}, metadata \rangle$ in the local database
- For the ID_{HI} and metadata recovery, the patient encrypts them with his public key. Then, the patient stores them in the blockchain. When the recovery, the patient searches the transaction that holds $enc[ID_{HI}]_{pat}$ and $enc[metadata]_{pat}$, and decrypts them with his private key.

3) ACCESS REQUEST WORKFLOW

Access request workflow is the procedure that a requestor seeks permission to obtain health information from a patient as illustrated in Figure 3(a). We assumed that the requestors know the ID_{HI} that they want to request through the patients and CA in advance. This workflow proceeds as follows:

- The requestor selects n of participants' addresses and public keys randomly from the PC

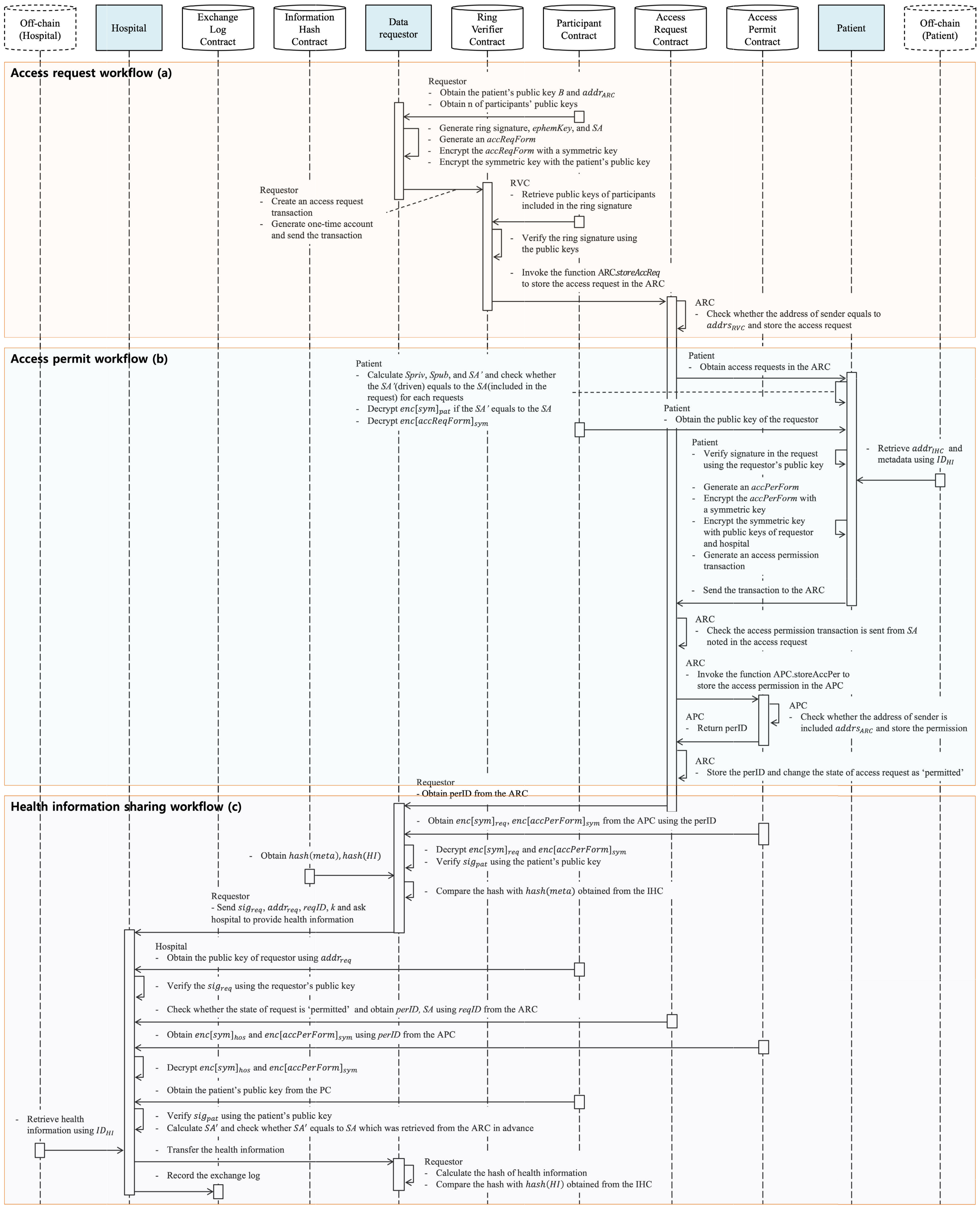
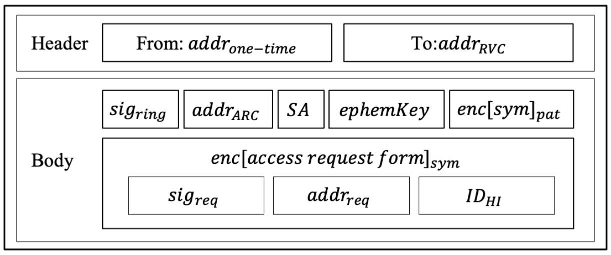
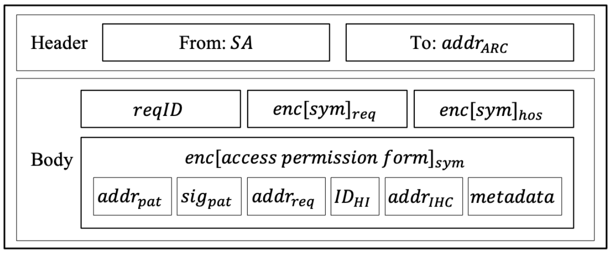

Off-chain (Hospital)
Hospital
Exchange Log Contract
Information Hash Contract
Data requestor
Ring Verifier Contract
Participant Contract
Access Request Contract
Access Permit Contract
Patient
Off-chain (Patient)
Access request workflow (a)
Requestor
- Obtain the patient's public key B and $addr_{ARC}$
- Obtain n of participants' public keys
- Generate ring signature, *ephemKey*, and *SA*
- Generate an *accReqForm*
- Encrypt the *accReqForm* with a symmetric key
- Encrypt the symmetric key with the patient's public key
Requestor
- Create an access request transaction
- Generate one-time account and send the transaction
RVC
- Retrieve public keys of participants included in the ring signature
- Verify the ring signature using the public keys
- Invoke the function ARC.*storeAccReq* to store the access request in the ARC
ARC
- Check whether the address of sender equals to $addrs_{RVC}$ and store the access request
Access permit workflow (b)
Patient
- Calculate *Spriv*, *Spub*, and *SA'* and check whether the *SA'*(driven) equals to the *SA*(included in the request) for each requests
- Decrypt $enc[sym]_{pat}$ if the *SA'* equals to the *SA*
- Decrypt $enc[accReqForm]_{sym}$
Patient
- Obtain access requests in the ARC
Patient
- Obtain the public key of the requestor
Patient
- Verify signature in the request using the requestor's public key
- Generate an *accPerForm*
- Encrypt the *accPerForm* with a symmetric key
- Encrypt the symmetric key with public keys of requestor and hospital
- Generate an access permission transaction
- Send the transaction to the ARC
- Retrieve $addr_{IHC}$ and metadata using ID_{HI}
ARC
- Check the access permission transaction is sent from *SA* noted in the access request
ARC
- Invoke the function APC.storeAccPer to store the access permission in the APC
APC
- Check whether the address of sender is included $addrs_{ARC}$ and store the permission
APC
- Return perID
ARC
- Store the perID and change the state of access request as 'permitted'
Health information sharing workflow (c)
Requestor
- Obtain perID from the ARC
- Obtain $enc[sym]_{req}$, $enc[accPerForm]_{sym}$ from the APC using the perID
- Obtain $hash(meta), hash(HI)$
- Decrypt $enc[sym]_{req}$ and $enc[accPerForm]_{sym}$
- Verify sig_{pat} using the patient's public key
- Compare the hash with $hash(meta)$ obtained from the IHC
Requestor
- Send sig_{req}, $addr_{req}$, *reqID*, *k* and ask hospital to provide health information
Hospital
- Obtain the public key of requestor using $addr_{req}$
- Verify the sig_{req} using the requestor's public key
- Check whether the state of request is 'permitted' and obtain *perID, SA* using *reqID* from the ARC
- Obtain $enc[sym]_{hos}$ and $enc[accPerForm]_{sym}$ using *perID* from the APC
- Decrypt $enc[sym]_{hos}$ and $enc[accPerForm]_{sym}$
- Obtain the patient's public key from the PC
- Verify sig_{pat} using the patient's public key
- Calculate *SA'* and check whether *SA'* equals to *SA* which was retrieved from the ARC in advance
- Retrieve health information using ID_{HI}
- Transfer the health information
- Record the exchange log
Requestor
- Calculate the hash of health information
- Compare the hash with $hash(HI)$ obtained from the IHC

FIGURE 3. Overview of workflow.

- The requestor retrieves the target patient's public key B and $addr_{ARC}$ from the PC
- The requestor generates the ring signature $sig_{ring} = \langle c_1, u_1, \ldots, u_n, addr_1, \ldots, addr_n \rangle$, $ephemKey = k \cdot G$ and $SA = hash(hash(k \cdot B) \cdot G)$ where k is a random number
- The requestor creates *accReqForm* containing the requestor's signature, requestor's ID, and the ID of health information, i.e., $\langle sig_{req}, ID_{req}, ID_{HI} \rangle$
- Then, the *accReqForm* is encrypted with a symmetric key
- The symmetric key is encrypted with the patient's public key
- The requestor creates an access request transaction containing the sig_{ring}, $addr_{ARC}$, SA, $ephemKey$, $enc[sym]_{pat}$, and $enc[accReqForm]_{sym}$ as shown in Figure 4(a)
- The requestor generates a one-time account and sends the access request transaction to the RVC with the sender as the one-time address
- The RVC obtains the public keys using addresses in $ring_{sig}$
- The RVC verifies the sig_{ring} using the *verifyRingSig* function
- The RVC invokes the *storeAccReq* of the ARC whose address corresponds to $addr_{ARC}$
- The ARC checks whether the sender address is included in $addrs_{ARC}$. Then, the ARC stores the access request.



(a) Access request transaction



(b) Access permission transaction

FIGURE 4. Transaction structures in the access request and permission.

4) ACCESS PERMIT WORKFLOW

Through the access permit workflow, a patient grants access to a requestor as shown in Figure 3(b). This workflow proceeds as follows:

- The patient obtains the access requests from the ARC to which he belongs
- The patient calculates stealth private key $sPriv = hash(privkey \cdot ephemKey) + privkey$, public key $sPub = sPriv \cdot G$, and address $SA' = hash(sPub)$. Then, the patient checks that SA' equals SA for all access requests
- If it equals, the patient decrypts $enc[sym]_{pat}$ with his private key and $enc[accReqForm]_{sym}$ with the symmetric key. Then, the patient obtains $\langle sig_{req}, ID_{req}, ID_{HI} \rangle$
- The patient retrieves the requestor's public key from the PC
- The patient verifies the sig_{req} using the requestor's public key
- The patient retrieves the $addr_{IHC}$ and metadata that matches the ID_{HI} and in the local database
- The patient generates an access permission form $\langle ID_{pat}, sig_{pat}, ID_{req}, ID_{HI}, addr_{IHC}, metadata \rangle$, and encrypts it with a symmetric key $enc[accPerForm]_{sym}$
- The patient encrypts the symmetric key with the public keys of requestor $enc[sym]_{req}$ and hospital $enc[sym]_{hos}$
- The patient carves an access permission transaction containing $reqID$, $enc[sym]_{req}$, $enc[sym]_{hos}$, and $enc[accPerForm]_{sym}$ as depicted in Figure 4(b). Then, the patient signs the transaction with $sPriv$ and sends it to his ARC (using the function *transferAccPer*)
- The ARC retrieves the SA corresponding to $reqID$ and checks that the sender's address is the SA
- The ARC invokes the function *storeAccPer* of APC as inputs $enc[accPerForm]_{sym}$, $enc[sym]_{req}$, $enc[sym]_{hos}$. If it returns $perID$, the ARC stores $perID$ of the access request and the *state* as 'permitted.'

5) HEALTH INFORMATION SHARING WORKFLOW

In this workflow, as shown in Figure3(c), requestors ask for health information from hospitals, and hospitals provide the requested health information to the requestors after checking patients' permissions. This workflow proceeds as follows:

- After noticing that the state of access request is changed, the requestor obtains $perID$ in his access request from the ARC
- Then, the requestor obtains $enc[accPerForm]_{sym}$ and $enc[sym]_{req}$ from the APC corresponding to the $perID$
- The requestor decrypts the $enc[sym]_{req}$ with his private key and $enc[accPerForm]_{sym}$ with the symmetric key, and obtains $\langle addr_{pat}, sig_{pat}, ID_{HI}, addr_{IHC}, metadata \rangle$
- The requestor verifies the sig_{pat} with the patient's public key
- The requestor obtains *hash(meta)* and *hash(HI)* from the IHC whose address is $addr_{IHC}$
- Then, the requestor calculates and compares the hash value of metadata with *hash(meta)*
- The requestor asks for health information to the hospital using the metadata while providing sig_{req}, $addr_{req}$,, $reqID$, and k where k is the random number that was used to generate *ephemKey* in the access request workflow
- Upon receiving the request, the hospital obtains the requestor's public key from the PC using $addr_{req}$

- The hospital verifies the sig_{req} using the requestor's public key
- The hospital checks whether the state of permission is permitted or not. If the state is 'permitted,' the hospital obtains *perID* and *SA* from the ARC using *reqID*
- The hospital retrieves $enc[sym]_{hos}$ and $enc[accPerForm]_{sym}$ from the APC using the *perID*
- Then, the hospital decrypts $enc[sym]_{hos}$ with the private key and $enc[accPerForm]_{sym}$ with the symmetric key, and obtains $\langle addr_{pat}, sig_{pat}, addr_{req}, ID_{HI}, addr_{IHC}, andmetadata\rangle$
- The hospital retrieves the patient's public key that matches with $addr_{pat}$ from the PC
- The hospital verifies the sig_{pat} using the public key
- To prevent stealing and utilizing $enc[accPerForm]$, the hospital calculates SA' using the patient's public key and k ($SA' = hash(hash(k \cdot B)) \cdot G + B$ where B is the public key of the patient). Subsequently, the hospital checks whether the SA' equals to SA retrieved from the ARC
- The hospital retrieves health information using ID_{HI} and delivers it to the requestor
- The hospital creates $\langle enc[log]_{sym}, enc[sym]_{pat}, enc[sym]_{hos}\rangle$ and stores them in the ELC using the function *storeLog*
- Upon receiving health information, the requestor calculates the hash of health information
- Then, the requestor compares the hash of health information with the $hash(HI)$ obtained from the IHC.

D. SOFTWARE COMPONENTS

MEXchange software components are structured into the presentation, business logic, and storage layers as depicted in Figure 5. The presentation layer consists of the user interface and controller for interacting with users directly. The controller forwards the user's request to the business logic layer and returns the results via the interface. The business logic layer connects the presentation layer with the storage layer. It consists of six components: blockchain, cipher, database, push, API manager, and scanner. The blockchain manager transmits transactions to the blockchain (e.g., access request and access permit transaction). The cipher manager encrypts data, and generates ring signature and stealth address. The database manager retrieves data in the local database. The push manager sends alarms to users. The API manager sends and receives requests to and from other participants such as CA and hospitals. The scanner finds access requests forwarded to the user through stealth address scanning. The storage layer manages the data required to operate MEXchange. It consists of the blockchain and local database. The blockchain contains six smart contracts and updates block data. The local databases store and manage private information such as health information, personal identity data.

E. IMPLEMENTATION

We implement MEXchange as a prototype based on the software components. The presentation layer is developed using HTML, CSS3, JavaScript. The business logic layer employs NodeJS and Web3.js to link private Ethereum blockchain. For the storage layer, we employ Go-Ethereum(v1.10.3) to construct the private Ethereum blockchain and use Solidity to implement six smart contracts with the solidity compiler v0.5.17+. The implemented smart contracts are deployed using Remix IDE. Additionally, we employed AES-256 scheme for symmetric encryption and secp256k1 elliptic curve for asymmetric encryption.

F. EVALUATION

We evaluate MEXchange in terms of both quantitative and qualitative aspects. In quantitative evaluation, we investigate the transaction latency and transaction per second for access request, request scan, and access permission which are key processes for exchanging health information in terms of the different number of transactions. We measure the transaction latency and transaction per second in the access request depending on the size of the ring signature (4, 8, and 12). Additionally, we compared the results of MEXchange with MedRec whose source code is available[1] for access request and permission. The reason for measuring only access request and permission is that MedRec does not utilize stealth address, so the request scan does not exist. We evaluate MedRec on a blockchain with the same configuration as MEXchange. We construct the private Ethereum blockchain using the PoW(Proof-of-Work) consensus algorithm for measurement. The configuration of the private network is shown in Table 1. The private blockchain consists of two machines. The first machine runs on a Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz, 32GB 2400 MHz DDR4 desktop. The second machine runs on an AMD Ryzen 5 3600X 6-Core Processor, 32GB 2667 MHz DDR4 desktop.

In the qualitative aspect, we adopted the method that evaluates blockchain solutions using non-functional requirements and properties as shown in Figure 6 [26]. The non-functional requirements specify criteria rather than a specific behavior to judge the operation of a system. The non-functional properties are performance categories for technology that stem from the requirements. Based on the evaluation method, we extract non-functional requirements of the HIE system from a report written by ONC [1]. Then, we derive key non-functional properties such as confidentiality, integrity, and availability from the requirements. Then, we evaluate MEXchange using the key non-functional properties listed.

G. THREAT MODELING

We adopted the threat modeling method suggested by Howard and Lipner [29] to identify security vulnerabilities of MEXchange. They proposed nine threat model steps: 1) define use scenarios; 2) gather a list of external dependencies; 3) define security assumptions; 4) create external security notes; 5) create Data Flow Diagrams (DFD) of the application being modeled; 6) determine threat types;

[1]https://github.com/mitmedialab/medrec

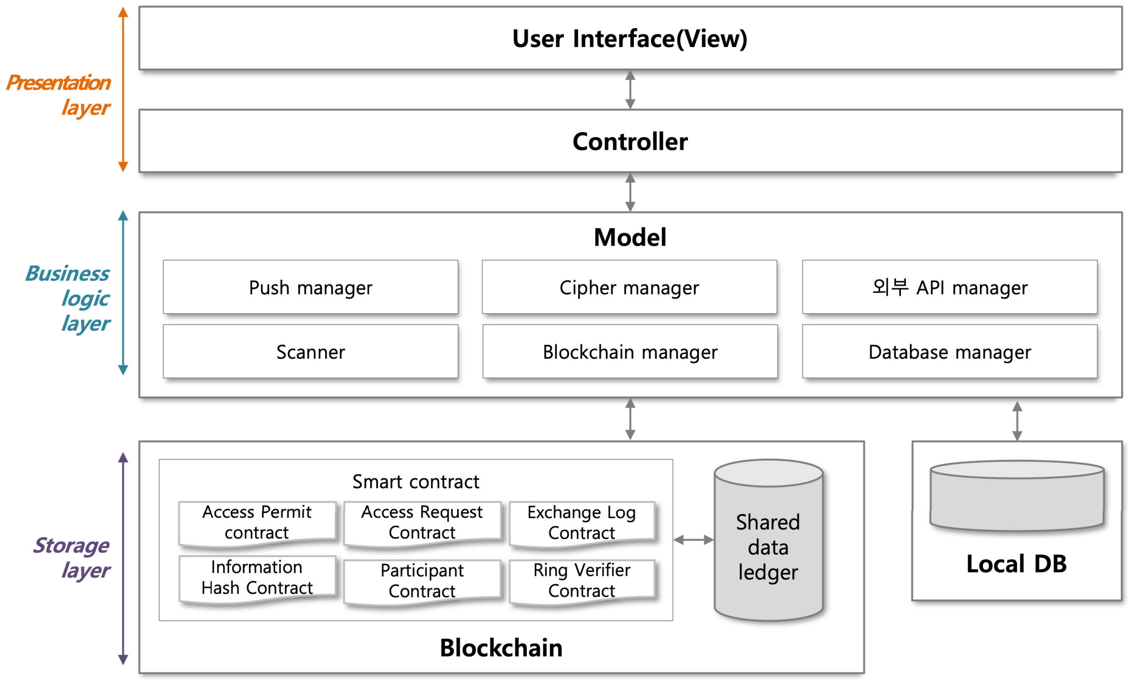


FIGURE 5. Software components for MEXchange.

TABLE 1. The private blockchain configuration.

Configuration	Value
homesteadBlock	0
eip150Block	0
eip155Block	0
eip158Block	0
byzantiumBlock	0
constantinopleBlock	0
difficulty	10
gasLimit	0x2100000
gasPrice	0

7) identify the threats to the system; 8) determine risk; 9) plan mitigations. Among the steps, we focused on from step 5 to step 7. Therefore, we derived DFD for exchanging health information in MEXchange. Subsequently, we determined threat types using STRIDE (Spoofing identity, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege) which is Microsoft's taxonomy. Finally, we identified the threats to MEXchange.

V. RESULTS

A. IMPLEMENTATION

Figure 7 shows the user interface of the prototype implemented. Figure 7(a) shows that a requestor generates an access request transaction and sends it to a patient through the blockchain. The requestor inputs $addr_{pat}$ and ID_{HI} in the interface. Then, the software generates ring signature, stealth address, and access request form and shows them through the interface. Figure 7(b) shows that the patient receives access requests delivered to him by alarm and permits access to his health information. When the patient clicks the permit button, the software generates an access permission form and sends it to the blockchain.

B. QUANTITATIVE EVALUATION

We measure transaction latency and throughput for three major operations: access request, request scan, and access permission. Furthermore, we compare the results of MEXchange with MedRec. The evaluation results represent the average of five independent trials.

In access request, we measure the transaction latency for access requests to be stored in the ARC, i.e., time taken for *verifyRingSig(), transferAccReq()*, and *storeAccReq()*. Subsequently, we calculate the TPS using the transaction latency. Figures 8 and 9 show the results, respectively. The transaction latency tends to grow as the number of transactions increases. The TPS tends to decrease and then becomes stable as the number of concurrent transactions increases. Regarding the ring size, transaction latency increases and the TPS decreases as the ring size increases. This is because the number of operations needed for ring signature verification increases as the ring size becomes bigger. Compared to MedRec, access requests in MEXChange take a long time and handle fewer transactions per second. When the TPS becomes stable, MedRec's throughput is about 1.2, 4, and 8 times that of ring sizes 4, 8, and 12. The reason is that MEXChange requires additional computational time in verifying ring signature to

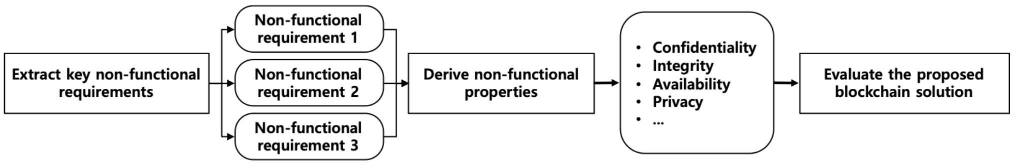
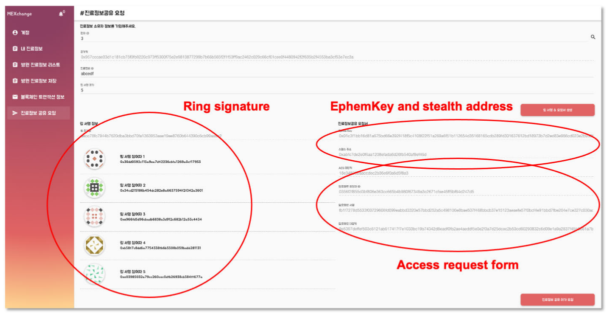
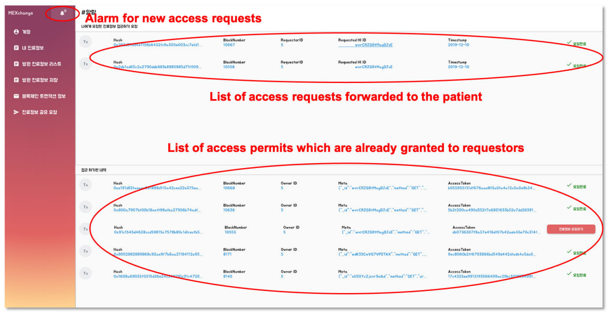

Extract key non-functional requirements
Non-functional requirement 1
Non-functional requirement 2
Non-functional requirement 3
Derive non-functional properties
• Confidentiality
• Integrity
• Availability
• Privacy
• ...
Evaluate the proposed blockchain solution

FIGURE 6. The schema for qualitative evaluation.



(a) Implementation result of Access request step


Alarm for new access requests
List of access requests forwarded to the patient
List of access permits which are already granted to requestors

(b) Implementation result of Access permission step

FIGURE 7. Prototype implementation results.

improve privacy. Moreover, MEXchange encrypts sensitive data to enhance security. This increases the transaction size and reduces the number of transactions processed in one block, which results in time delay and lower TPS.

In request scan measurement, we defined the transaction latency as the time taken to find access requests and derive *sPriv* using the *ephemKey* and *SA* contained in the access requests. Figure 10 and 11 show the results of transaction latency and TPS. In this result, the time taken tends to increase linearly as the number of transactions scanned increases, which yields an almost constant TPS. Furthermore, the access request scan time takes less time compared to access requests and access permission. This is because the scanning is done on the local machine, so no latency occurs during transaction propagation and processing.

In access permission measurement, the transaction latency is defined as the time taken to broadcast access permission transactions and store them in the APC, i.e., time taken for *transferAccPer()* and *storeAccPer()*. Figure 12 shows the transaction latency of access permission. Subsequently, we calculate the TPS using the measured transaction latency as shown in Figure 13. The transaction latency tends to grow as the number of transactions increases. In the TPS, the throughput of MEXchange and MedRec increases until the number of transactions is 200 and 300, respectively, and then decreases and becomes stable. This is because transactions are contained in one block until the number of transactions is 200 and 300, respectively. That is, although the processing time is almost the same, the number of transactions processed increases. Therefore, the throughput increases. After that, the number of blocks needed to process transactions increases, and throughput decreases and becomes stable. When the TPS is stable, the throughput of MEXchange is about 50% of MedRec. This is because MEXchange stores the data as encrypted in the blockchain, which increases the transaction size and lowers the number of transactions contained in one block.

C. QUALITATIVE EVALUATION

We extracted four key non-functional requirements and identified eight key non-functional properties from the requirements.

- *Protect privacy and security in all aspects of interoperability and respect individual preferences.* The requirement insists stakeholders propose effective protection for health information against problems arising from HIE to assure public trust that health information is safe and secure. Also, it is essential to inform patients of details in which their data is utilized. We identified five key non-functional properties from the requirement, which are *availability, confidentiality, integrity, privacy, transparency*.
- *Verifiable identity and authentication of all participants.* Interoperable HIE methods must authenticate participants so participants do not disguise or impersonate others. We identified *authenticity* as one of the key properties in this requirement.
- *Build upon the existing health IT infrastructure.* The requirement requests stakeholders to improve interoperability based on their existing IT infrastructure. We identified the *ease of integrating new data providers* as one of the key properties.
- *Build a culture of electronic access and use.* Interoperable HIE methods should be open to anyone, handling large and growing electronic health information to reduce the digital divide. We identified *scalability* as the key property.

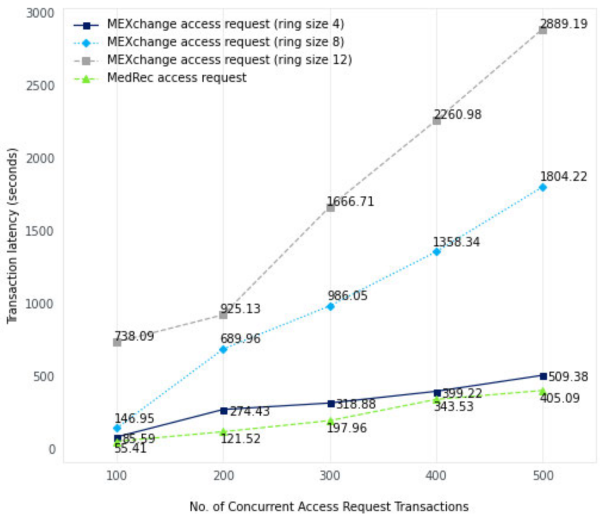


FIGURE 8. Transaction latency for access request.

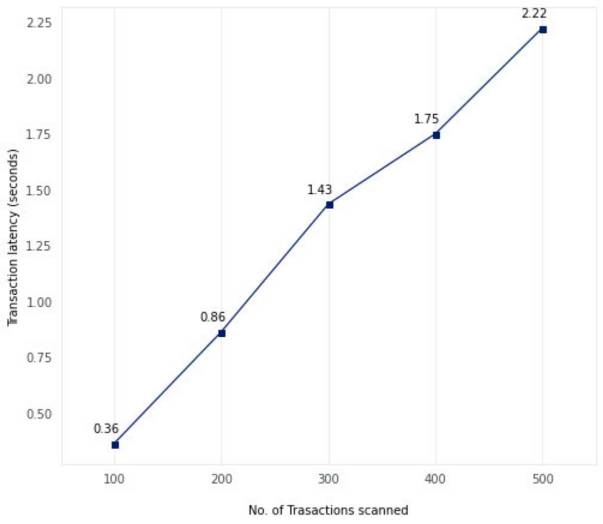


FIGURE 10. Transaction latency for request scan.

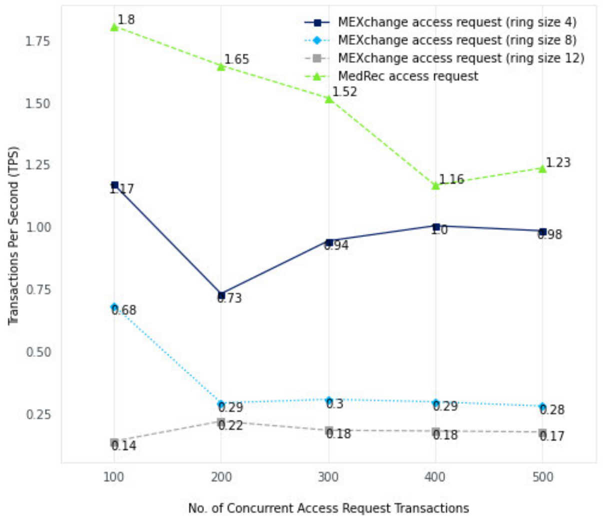


FIGURE 9. Transaction per second for access request.

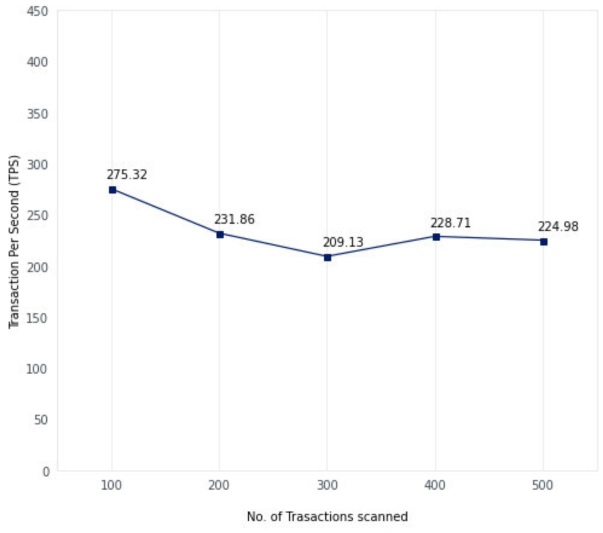


FIGURE 11. Transaction per second for request scan.

As a result, we identified eight key non-functional properties from the requirements. Using the key properties we evaluate MEXchange qualitatively as below.

- Authenticity. MEXchange authenticates new users through CA and manages them in the blockchain. Furthermore, MEXchange verifies that the ring signature consists only of the authorized participants to prevent unauthorized participants from engaging in exchange.
- Availability. In MEXchange, metadata and permission are stored and managed in the blockchain, which assures data integrity and enables participants to exchange health information unless the blockchain is compromised. When a hospital's database is attacked, participants cannot exchange health information managed by the hospital. However, exchanging health information held by other hospitals is possible.
- Confidentiality. With metadata and permission stored in the blockchain as encrypted form, only authorized participants can decrypt and access them. Also, hospitals provide requestors with health information upon confirming that the requestors were granted access.
- Ease of integrating new data providers. MEXchange does not store health information in the blockchain but employs databases governed by the hospitals. Therefore, when a hospital becomes a participant, the hospital does not need to transfer health information elsewhere.
- Integrity. MEXchange manages data through the blockchain, making it impossible for attackers to

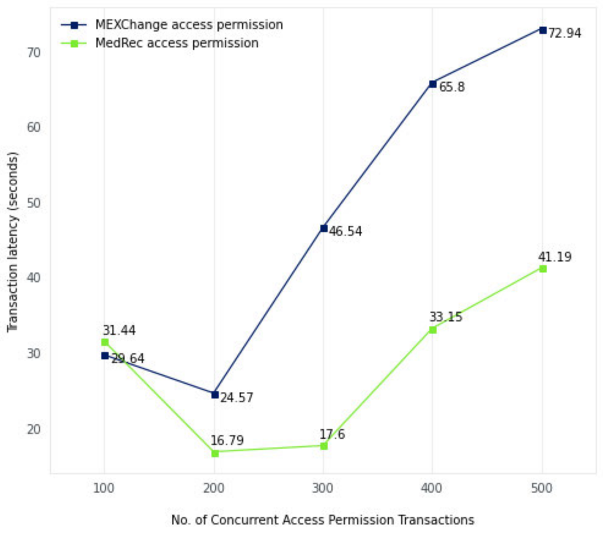


FIGURE 12. Transaction latency for access permission.

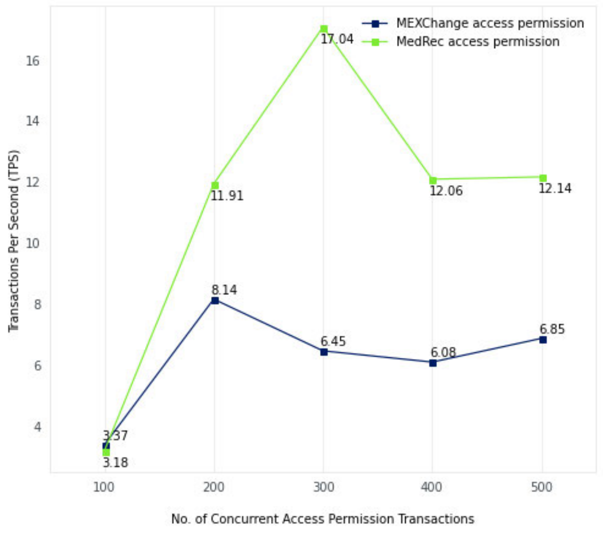


FIGURE 13. Transaction per second for access permission.

counterfeit data. Moreover, it stores the hash value of health information and metadata in the blockchain. Therefore, participants can confirm the integrity of data by comparing hash values.

- Privacy. MEXchange enhances privacy by obscuring sender and receiver addresses using the ring signature and stealth address to prevent the inference problem. MEXchange also encrypts data stored in the blockchain and prevents sensitive information from leaking. Therefore, the data publicly visible in MEXchange is $addr_{one-time}$, $addr_{RVC}$, sig_{ring}, $addr_{ARC}$, SA, $ephemKey$, and $reqID$, except encrypted data. It is hard to infer senders and receivers with $addr_{one-time}$, SA, and $ephemKey$, because they are transient data and are not linked to senders and receivers directly. Although sig_{ring} and $addr_{ARC}$ contain information related to senders and receivers, it is hard to infer senders and receivers using the data. The sig_{ring} includes the addresses of a sender and participants randomly selected; thus, it is difficult to de-anonymize senders compared to blockchain-based HIE without the ring signature. The address of the ARC refers to the access request storage where the patient monitors. Therefore, malicious participants could infer some participants using $addr_{ARC}$. However, it is difficult to specify a receiver using $addr_{ARC}$ because some of the other patients also are included in the same access request storage. The remaining $addr_{RVC}$ and $reqID$ are unrelated data to the senders and receivers.
- Scalability. A disadvantage of the blockchain is being unable to handle a large volume of transactions since it continuously shares and verifies transactions among participants. Thus, MEXchange cannot handle large volumes of transactions quickly. Moreover, MEXchange has overhead due to the ring signature and stealth address, although it attempts to reduce the scan time by dividing participants into clusters.
- Transparency. MEXchange allows patients to audit and trail sharing logs after exchanging health information with other participants. Therefore, the patients can obtain the logs from the blockchain and check the exchange history of their health information.

D. THREAT MODELING

This section describes the results of threat modeling. First, we derived the DFD of MEXchange as shown in Figure 14. Based on the DFD, we identified threats to the MEXchange by referencing the Microsoft threat tool.[2] As a result, we identified 232 threats, with 33 for S and 54 for T, 27 for R, 87 for I, 2 for D, and 29 for E by category as explained in Table 2. Most of the spoofing threats involve the cases that malicious attackers gain unauthorized privileges on the business logic layer through participants' machines. The tampering threats are related to changing the databases managed by the patient, hospital, and certificate authority through SQL and malicious input injection. The repudiation threats involve denying the actions on the business logic layer and database managed by patients when storing metadata. In terms of information disclosure, there are threats of obtaining sensitive data from participants' devices or sniffing the device's traffic. In terms of denial of service, there are threats of disrupting the participant registration and health information sharing service by attacking the CA and hospitals. Threats of elevation of privileges include obtaining privileges through attacks on participants' devices and unauthorized access to their database.

VI. DISCUSSION

A. COMPARATIVE ANALYSIS

We compare MEXchange with other HIE systems such as Ancile, FHIRChain, IHE XDS, and MedRec. Results

[2]https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

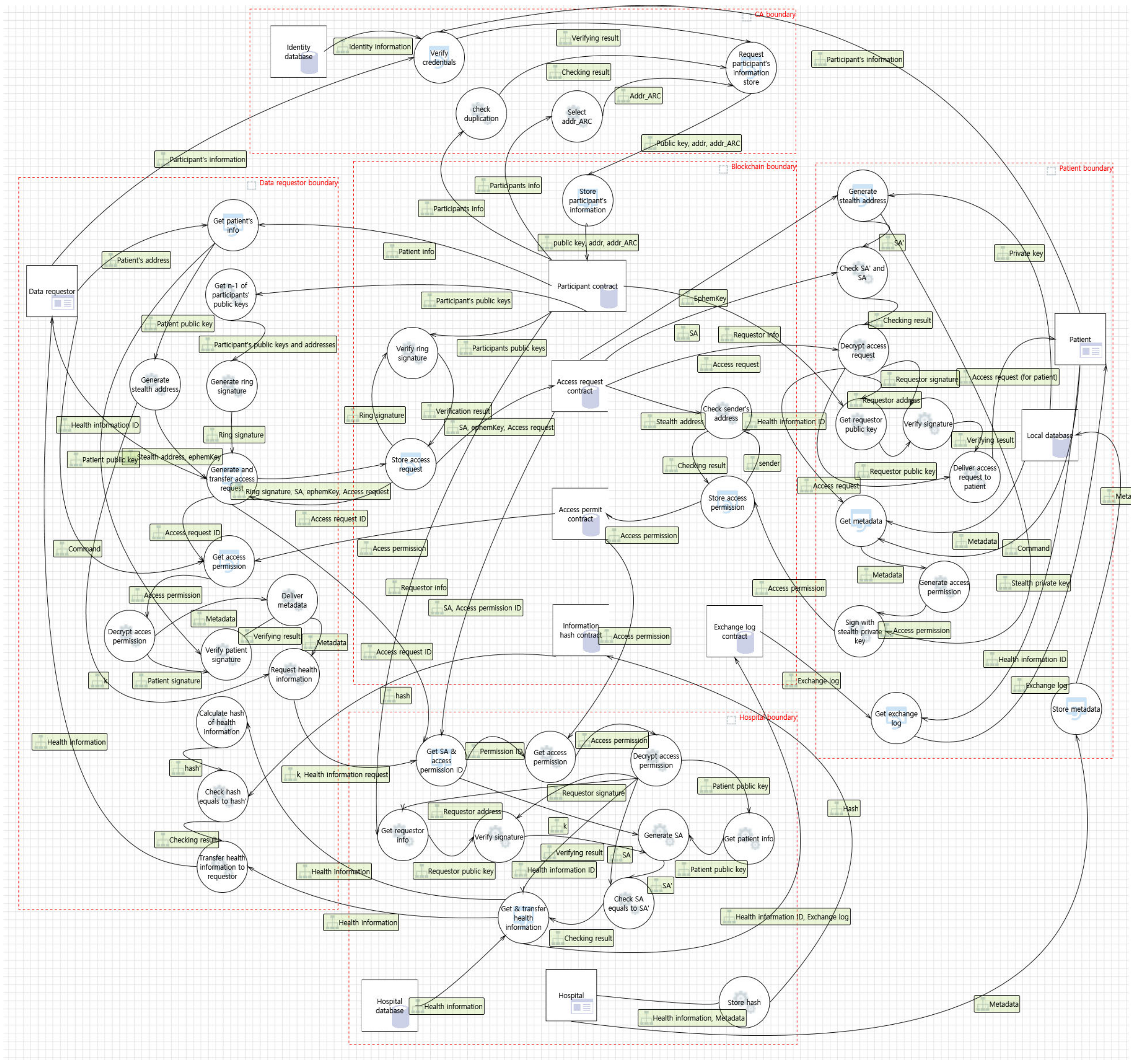


FIGURE 14. Data flow diagram for MEXchange.

TABLE 2. STRIDE threat modeling result.

Category	S	T	R	I	D	E
Frequency	33	54	27	87	2	29
Ratio(%)	14.2	23.3	11.7	37.5	0.1	12.5

from the comparative analysis are summarized in Table 3. Ancile, FHIRChain, and MedRec are the systems employing the blockchain to manage metadata and access permissions. IHE XDS is a document-based standard architecture designed for EHR sharing system [30]. In XDS, data providers register their health information in the document repository. Subsequently, the document repository submits the metadata of documents to the system called document registry.

- *Authenticity.* Ancile, FHIRChain, XDS, and MedRec authenticate new users through CA. After authentication, each system manages users' identity data in the blockchain. In XDS, a system called Patient Identity Source manages identity data.
- *Availability.* Ancile, FHIRChain, and MedRec are similar to MEXchange in that they employ the blockchain and the hospitals' databases. However, Ancile utilizes proxy nodes to reduce the burden of key sharing. Thus, participants cannot exchange health information when proxy nodes are compromised. In XDS, metadata exchange is impossible if the document registry or repository is compromised.
- *Confidentiality.* FHIRChain encrypts and stores data in the blockchain so only authorized participants can decrypt and access the data. Ancile also manages

TABLE 3. Results of comparative analysis.

Aspect	MEXchange(ours)	Ancile[7]	FHIRchain[6]	IHE XDS[30]	MedRec[5]
Authenticity	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕
Availability	⊕ ⊕ ⊕	⊕ ⊕	⊕ ⊕ ⊕	⊕	⊕ ⊕ ⊕
Confidentiality	⊕ ⊕ ⊕	⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕
Ease of integrating new data providers	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕ ⊕
Integrity	⊕ ⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕	⊕	⊕ ⊕ ⊕
Privacy	⊕ ⊕ ⊕	⊕	⊕	⊕ ⊕	⊕
Scalability	⊕	⊕ ⊕	⊕ ⊕	⊕ ⊕ ⊕	⊕ ⊕
Transparency	⊕ ⊕ ⊕	⊕	⊕	⊕	⊕

some data as an encrypted form but does not encrypt data such as database addresses and permission levels. XDS allows only authorized participants to access the document registry and repository. MedRec does not suggest a method or workflow for encrypting data stored in the blockchain.

- *Ease of integrating new data providers.* Ancile, FHIRChain, and MedRec leave management of health information to the hospitals. A new hospital needs not to transfer health information elsewhere. However, in XDS, the hospitals register their health information in the document repository to share, which is additional work to the data providers.
- *Integrity.* Ancile and MedRec manage metadata and permission in the blockchain and store the hash value of health information in the blockchain. Similarly, FHIRChain stores metadata in the blockchain to assure data integrity. However, it does not store the hash value of health information in the blockchain. In XDS, the document registry and repository are a single point where data can be falsified relatively easily by attackers. Thus, the participants cannot check the integrity of health information even data is manipulated.
- *Privacy.* The inference problem can arise in Ancile, FHIRChain, and MedRec because senders and receivers are visible to any participants. Moreover, in MedRec, attackers can obtain personal information easily from the blockchain because it does not encrypt data. In XDS, personal information is less likely to leak because only permitted participants to access the system and obtain data. However, there is still a possibility of information leakage from entities that operate the XDS system.
- *Scalability.* A disadvantage is that it is hard to handle a large volume of transactions. Thus, Ancile, FHIRChain, and MedRec cannot handle large transactions quickly. Compared to the blockchain-based HIE, XDS copes with the increasing volume of transactions through scale-up and scale-out flexibly because it is a centralized system managed by operators.
- *Transparency.* Ancile, FHIRChain, XDS, and MedRec do not mention a method to record exchange history for patients to audit and trail.

B. CONTRIBUTIONS AND LIMITATIONS

Privacy and security concern is major challenge for HIE and must be considered [3]. MEXchange enhances privacy by adopting the ring signature and stealth address to prevent the inference problems that arise from existing blockchain-based HIE. It encrypts data stored in the blockchain to increase confidentiality, which strengthens security. As we have suggested general smart contracts to utilize the ring signature and stealth address, MEXchange can be applied to any blockchain platform that supports smart contracts. With an overall workflow in health information sharing workflow and implemented prototype with Ethereum, we show that blockchain-based HIE with the ring signature and stealth address can be used in healthcare.

Nonetheless, MEXchange has several limitations. MEXchange takes a long time to process access requests since ring signature verification requires many elliptic curve operations. As an alternative, the ring signature based on RSA, which shows better performance than ECC in encryption time, can be considered as future research. Furthermore, other blockchain and consensus algorithms such as Hyperledger Fabric, Corda, PoS(Proof-of-Stake), and DPoS(Delegated-Proof-of-Stake) which show better scalability than PoW can be considered. Second, we did not suggest the guidelines for the proper ring and cluster size. The larger the ring size is, the lower the possibility of inference. However, the time to generate and verify the ring signature takes more. Also, as the number of patients included in the cluster (cluster size) increases, inference becomes more difficult, but the number of transactions to be scanned increases. Therefore, future studies are needed to determine the appropriate ring and cluster size. Another limitation is that managing sensitive data such as private key, metadata, and ID_{HI} can be a burden for the patients. Such sensitive data should be managed safely because it is used for requesting and permitting health information exchange. Recently, some tools for blockchain key management have been developed for blockchain application users. Therefore, future research to manage sensitive data can be considered when applying blockchain in the healthcare field. In addition, we did not consider the case that participants exchange wrong information. For example, when the patient delivers a wrong symmetric key to the requestor, a method for supporting the requestor to re-exchange the symmetric key is needed. In this case, we could consider 3-way Diffie–Hellman key exchange as a solution. Therefore, we plan to proceed with further research for dealing with such a case. Finally, this research does not consider the different formats of health information. Therefore, future studies should consider the limitations mentioned above.

VII. CONCLUSION

This paper presented a blockchain-based HIE framework, MEXchange, that leverages the ring signature and stealth address to prevent the inference problem. We implemented the proposed framework as a prototype and evaluated it in the quantitative and qualitative aspects. Furthermore, we analyzed possible threats on MEXchange. Conclusively, MEXchange has disadvantages of increasing transaction latency and lower TPS compared to the existing blockchain-based HIE, however, it has advantages in availability, confidentiality, integrity, privacy, and transparency. In particular, MEXchange shows strength in the privacy aspect by preventing the inference problem. MEXchange has several limitations discussed in Section VI. As future works, we plan to increase the scalability, determine appropriate ring and cluster size, and consider different health information formats. To improve scalability, we will evaluate MEXchange on other blockchain platforms which reduce the time taken to generate and verify the ring signature. We will attempt to find out the proper ring size and cluster, which balances scalability and security. In addition, we will shorten the time to scan transactions, which reduces exchanging time. Also, additional measures are needed to defend against threats that arise in the local function and database. Lastly, we will make it possible to exchange health information between hospitals that use different health information formats. Complementing the limitations, we expect that MEXchange lowers barriers to the actual application of blockchain-based HIE systems by mitigating concerns over privacy and security.

REFERENCES

[1] K. DeSalvo, "Connecting health and care for the nation: A shared nationwide interoperability roadmap draft version 1.0," in *Office of the National Coordinator for Health Information Technology*. Philadelphia, PA, USA: American College of Physician, 2015.

[2] G. J. Kuperman, "Health-information exchange: Why are we doing it, and what are we doing?" *J. Amer. Med. Inform. Assoc.*, vol. 18, no. 5, pp. 678–682, Sep. 2011.

[3] C. S. Kruse, V. Regier, and K. T. Rheinboldt, "Barriers over time to full implementation of health information exchange in the united states," *JMIR Med. Informat.*, vol. 2, no. 2, p. e26, Sep. 2014.

[4] K. B. Eden, A. M. Totten, S. Z. Kassakian, P. N. Gorman, M. S. McDonagh, B. Devine, M. Pappas, M. Daeges, S. Woods, and W. R. Hersh, "Barriers and facilitators to exchanging health information: A systematic review," *Int. J. Med. Informat.*, vol. 88, pp. 44–51, Apr. 2016.

[5] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Aug. 2016, pp. 25–30.

[6] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, "FHIRChain: Applying blockchain to securely and scalably share clinical data," *Comput. Struct. Biotechnol. J.*, vol. 16, pp. 267–278, Jul. 2018.

[7] G. G. Dagher, J. Mohler, M. Milojkovic, and P. B. Marella, "Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology," *Sustain. Cities Soc.*, vol. 39, pp. 283–297, May 2018.

[8] Q. Xia, E. B. Sifah, A. Smahi, S. Amofa, and X. Zhang, "BBDS: Blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 2, p. 44, 2017.

[9] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, and J. He, "Blochie: A blockchain-based platform for healthcare information exchange," in *Proc. IEEE Int. Conf. smart Comput. (SmartComp)*, Jun. 2018, pp. 49–56.

[10] I. Abu-Elezz, A. Hassan, A. Nazeemudeen, M. Househ, and A. Abd-Alrazaq, "The benefits and threats of blockchain technology in healthcare: A scoping review," *Int. J. Med. Informat.*, vol. 142, Oct. 2020, Art. no. 104246.

[11] A. Hasselgren, K. Kralevska, D. Gligoroski, S. A. Pedersen, and A. Faxvaag, "Blockchain in healthcare and health sciences—A scoping review," *Int. J. Med. Informat.*, vol. 134, Feb. 2020, Art. no. 104040.

[12] A. Shahnaz, U. Qamar, and A. Khalid, "Using blockchain for electronic health records," *IEEE Access*, vol. 7, pp. 147782–147795, 2019.

[13] Y. Zhuang, L. Sheets, Z. Shae, J. J. Tsai, and C.-R. Shyu, "Applying blockchain technology for health information exchange and persistent monitoring for clinical trials," in *Proc. AMIA Annu. Symp.*, 2018, p. 1167.

[14] T. Nugent, D. Upton, and M. Cimpoesu, "Improving data transparency in clinical trials using blockchain smart contracts," *FResearch*, vol. 5, p. 2541, Oct. 2016.

[15] S. Badr, I. Gomaa, and E. Abd-Elrahman, "Multi-tier blockchain framework for IoT-EHRs systems," *Proc. Comput. Sci.*, vol. 141, pp. 159–166, Jan. 2018.

[16] V. Patel, "A framework for secure and decentralized sharing of medical imaging data via blockchain consensus," *Health Inform. J.*, vol. 25, no. 4, pp. 1398–1411, 2019.

[17] ETRI. (2018). *Report of ETRI: Where the Blockchain*. [Online]. Available: https://library.etri.re.kr/service/rsch/issue-report/down.htm?view=open%&id=626

[18] M. Fleder, M. S. Kester, and S. Pillai, "Bitcoin transaction graph analysis," 2015, *arXiv:1502.01657*.

[19] A. Gaihre, Y. Luo, and H. Liu, "Do bitcoin users really care about anonymity? An analysis of the bitcoin transaction graph," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 1198–1207.

[20] MedRec. (2016). *Medrec Technical Documentation*. [Online]. Available: https://medrec.media.mit.edu/technical/

[21] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology—ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Germany: Springer, 2001, pp. 552–565.

[22] N. T. Courtois and R. Mercer, "Stealth address and key management techniques in blockchain systems," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 559–566.

[23] J. Liu, X. Li, L. Ye, H. Zhang, X. Du, and M. Guizani, "BPDS: A blockchain based privacy-preserving data sharing for electronic medical records," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[24] A. A. Alomar, M. Z. A. Bhuiyan, and 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, Jun. 2019.

[25] A. Zhang and X. Lin, "Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain," *J. Med. Syst.*, vol. 42, no. 8, p. 140, 2018.

[26] X. Xu, I. Weber, and M. Staples, *Architecture for Blockchain Applications*. Berlin, Germany: Springer, 2019.

[27] Ethereum. (2020). *Ethereum White Paper*. [Online]. Available: https://ethereum.org/en/whitepaper/

[28] Y. F. Chung, Z. Y. Wu, and T. S. Chen, "Ring signature scheme for ECC-based anonymous signcryption," *Comput. Standards Interfaces*, vol. 31, no. 4, pp. 669–674, Jun. 2009.

[29] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, Jun. 2006.

[30] IHE. (2020). *It Infrastructure (ITI) Technical Framework*. [Online]. Available: https://www.ihe.net/uploadedFiles/Documents/ITI/IHE_ITI_TF_Vol1.pdf

DEOKSANG LEE received the M.S. degree from the Department of Industrial & Management Engineering, Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 2020, where he is currently pursuing the Ph.D. degree with the Department of Industrial & Management Engineering. He has participated in several research project funded by the Information Technology Research Center (ITRC), South Korea, Samsung Electronics, Pohang Iron and Steel Corporation, and Samsung Construction & Trading Corporation. His research interests include blockchain, deep learning, process mining, and data science.

MINSEOK SONG (Member, IEEE) received the Ph.D. degree from the Department of Industrial & Management Engineering, Pohang University of Science and Technology (POSTECH), in 2006. He is currently an Associate Professor with the Department of Industrial & Management Engineering, POSTECH. Prior to this, he stayed with the Information Systems Group, Department of Industrial Engineering & Innovation Sciences, Eindhoven University of Technology, as a Postdoctoral Researcher, from 2006 to 2009. He was an Assistant Professor/an Associate Professor with the Ulsan National Institute of Science and Technology (UNIST) from 2010 to 2015. He has published more than 100 scientific papers in several top-level venues, such as *Decision Support Systems*, *Information Systems*, *Journal of Information Technology*, *International Journal of Medical Informatics*. His research interests include business process management, process mining, business analytics, simulation, and health information systems.