IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 May 2023, accepted 18 June 2023, date of publication 21 June 2023, date of current version 27 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3288332

RESEARCH ARTICLE

Secure and Lightweight Blockchain-Enabled Access Control for Fog-Assisted IoT Cloud Based Electronic Medical Records Sharing

SOMCHART FUGKEAW, (Member, IEEE), LEON WIRZ, AND LYHOUR HAK
Sirindhorn International Institute of Technology, Thammasat University, Pathum Thani 12120, Thailand

Corresponding author: Somchart Fugkeaw (somchart@siit.tu.ac.th)

This work was supported by the Sirindhorn International Institute of Technology (SIIT) Young Researcher Grant under Contract SIIT2019-YRG-SF02.

ABSTRACT As for the advancement of IoT and cloud computing in healthcare, outsourcing encrypted Electronic medical records (EMRs) created by the aggregation of medical treatment applications and health data collected from IoT devices enables high accessibility, effective collaboration, and zero computational operation cost. Current applications and research works generally concern the privacy of the finest EMRs that are encrypted with secure and lightweight cryptographic protocols before they are outsourced to the cloud. However, this process does not consider the security and privacy of the data collected by IoT devices, where the data being transferred can be leaked before they are aggregated. Furthermore, existing IoT-cloud based access control solutions have not addressed the outsourced encryption, privacy of IoT data transmission and aggregation, and the policy update of the EMRs in an integrated manner. In this paper, we propose an access control scheme called LightMED which provides secure, fine-grained, and scalable EMR sharing in a cloud-based environment integrated with fog computing, CP-ABE, and blockchain technology. We propose a secure IoT data transmission and aggregation method based on lightweight encryption and digital signing. At the core, we introduce outsourced encryption with a privacy-preserving access policy scheme and an outsourced encryption and decryption algorithm leveraged by the collaboration between fog nodes and blockchain. In addition, we introduce a novel lightweight policy update algorithm to enable the data owners of EMRs to effectively manage their policies in a secure and effective manner. Finally, we performed the comparative analysis to illustrate the computation cost and conducted experiments to evaluate the performance of our scheme and related works. The experimental results showed that our scheme outperformed existing works since it yielded least processing cost of both encryption and decryption at end-users' devices, which demonstrates the higher efficiency and practicality of our scheme.

INDEX TERMS IoT, electronic medical records, access control, cloud, fog computing, CP-ABE, blockchain.

I. INTRODUCTION

Medical data is considered a type of sensitive data which must be protected thoroughly in all aspects. Electronic Medical Records (EMRs) have particularly been watched closely to prevent any data loss or leakages to unwanted parties. One of the main sources that generate fully useable and efficient EMRs are IoT Devices which have sufficient capabilities of their sensors to monitor, detect, collect, and transmit patients' health conditions to belonging devices' owners or destinations over a period of time.

Currently, IoT integrated with cloud computing has played a significant role in the healthcare industry where generated healthcare data or applications are resorted to the cloud. The flexibility in accessing EMRs through the IoT or mobile devices is also desirable for today's healthcare industries. Patients or doctors can request that data easily and quickly once their EMRs were generated. However, the concern of

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Ali.

data privacy in the cloud is still troublesome for the whole healthcare industry. Consequently, it is challenging to regulate secure IoT data aggregation and fine-grained access to the outsourced EMRs data.

Specifically, ciphertext-policy attribute-based encryption (CP-ABE) [1] is a suitable access control solution for outsourced data because it provides both encryption features

and access control enforcement. In CP-ABE, the data owner can specify an access policy constructed from the set of attributes which is logically modeled through the logical gates AND, OR, and MofN to encrypt the data. Only the satisfied secret key can make decryption to the requested ciphertext.

For a practical deployment of CP-ABE in the cloud-based IoT healthcare environment, there are several issues to address. First, the cryptographic cost of CP-ABE is computationally expensive as it deals with pairing and exponentiation operations. Especially, the encryption and decryption of CP-ABE are not suitable for the data owner or the data user that uses resource-constraint devices such as mobile or IoT devices to encrypt or decrypt the ciphertext. Recent works [2], [3], [4], [5], [6] proposed CP-ABE decryption outsourcing to enable the user to efficiently decrypt the ciphertext. Despite the availability of several studies recommending data users' participation in partial CP-ABE decryption, the majority of these studies do not support outsourced encryption. Furthermore, lightweight encryption is unnecessary unless the data owner intends to encrypt the data through a resource-constrained device.

Second, the privacy of the secret key stored in IoT devices or user terminals that can be accessed by multiple users in each hospital department. The possibility of key leakage is high. Previous studies [7], [8], [9] proposed a traceable ABE to support the key leakage problem. However, such a technique is not a preventive solution.

Third, the cost of policy updates in the evolving access control for healthcare IoT data is high. To deal with the policy update, the ciphertext encrypted by the old policy needs to be re-encrypted and the user key may need to be re-generated. These costs are non-trivial. Previous studies [10], [11], [12], [13] proposed the outsourced policy update technique based on either ciphertext update [11], [12], [13] or proxy re-encryption (PRE) [10]. However, the cost of ciphertext update is still based on the partial computation of CP-ABE while the PRE relies on the proxy which is vulnerable to a single point of failure.

In addition to the shortfalls of CP-ABE mentioned above, the privacy of health data collected from IoT devices before they are aggregated is crucial for privacy-preserving EMRs access control. Also, most existing privacy-preserving access control for outsourced data overlooked the full-fledged access control functions including authentication and auditing in the integrated solution. Finally, most works implemented the invocation and processing of cryptographic operations using the proxy or outsourced agents. This is vulnerable to trust and computation bottlenecks.

To the best of our knowledge, there are no works that resolve all the above issues in an integrated manner. To this end, we propose a scheme called LightMED to offer secure, lightweight, and efficient privacy-preserving IoT-based EMRs sharing with policy update functionality. The proposed scheme leverages blockchain technology to support decentralized access authentication control and auditing. This entails secure healthcare data aggregation based on IoT data encryption and other treatment records. We also introduce the signing algorithm to support the source authentication of various records collected from IoT devices which is not supported by most of the existing CP-ABE schemes. To improve the communication as well as the computation cost of data outsourcing, multiple fog nodes are designed to perform CP-ABE encryption and liaise with the cloud server and blockchain. To allow the fog nodes to encrypt the data without the disclosure of policy content, we proposed an attribute tree-based encryption to preserve the privacy of the access policy while it is sent from the data owner to the fog node. For the decryption segment, we improve the secret key computation technique proposed in [2] by partitioning the random elements used to compute the key. With this improvement, our scheme does not require the secret key to be stored in the end user's device and the cost of re-encryption performed by the fog nodes is significantly reduced.

In summary, the contributions of this article are described as follows:

1. We proposed LightMED, a scheme that provides both lightweight outsourced encryption and decryption in the fog node environment. This scheme proposes a new way of dual encryption based on the Advanced Encryption Standard (AES-256) and Ciphertext-Policy Attribute-based-Encryption (CP-ABE) in a very lightweight algorithm with the assisted fog node to outsource the cryptographic processing cost to and from the cloud.
2. Our proposed scheme does not require the secret key to be stored on the end user's device. This prevents key leakage problems.
3. We devised a lightweight policy update algorithm to support efficient policy update management for outsourced EMRs. The data owners in our system can update the policy while the cost of ciphertext re-encryption is offloaded to the fog node.
4. We proposed an algorithm to encrypt the data from IoT devices and another to decrypt the data after receiving it from IoT devices in a secure and lightweight manner. In addition, we introduced a secure IoT data aggregation with source authentication before generating EMRs with our lightweight encryption and digital signing. These two algorithms can prevent any data leakages of patients once the sensor collects and transfers to the aggregation terminal or data collection.
5. We utilized blockchain technology to enable decentralized access control and authentication, establish

ciphertext indexing, and store certain cryptographic parameters to facilitate efficient ciphertext decryption. We bundled the ciphertext of the encrypted symmetric key, the encrypted patient's data, and a random value to be stored for ciphertext indexing and ciphertext decryption. To support these functions, we have introduced a set of smart contracts to flexibly automate the algorithms and all transaction records are recorded in an immutable manner in the blockchain.

6. We performed comparative analysis and conducted experiments to substantiate the efficiency of our scheme. We compared our work with existing papers to demonstrate the superiority of our proposed scheme and related works.

The residual part of this paper is organized as follows. Section II describes related works. Section III illustrates the technical and theoretical background of bilinear maps and blockchain. Section IV presents our proposed scheme and system overview. Section V represents the security analysis. Section VI shows functionality comparison, computation cost analysis, and performance evaluation. Section VII provides discussion of the experimental findings. Finally, the conclusion and future work are discussed.

II. RELATED WORKS

Several CP-ABE schemes have been proposed to tackle secure and fine-grained data sharing in the IoT cloud setting. Previous studies focus on the design and implementation of lightweight CP-ABE decryption through the outsourcing model.

In [14] the authors have outsourced the signing and verification, and decryption process of the data collected from IoT devices to a fog layer. In this scheme, the length of signature in their KPABS is constant. Hence, it is independent of the number of attributes used in signature. Yet the computational heavy process of encryption remains fully done on the IoT side.

In [3], the authors proposed a CP-ABE based access control for securing IoT data in a cloud environment. In this scheme, the cloud server is offloaded to perform a significant part of the partial CP-ABE decryption step via a user-specific transformation key. The transformation key is closely associated with the keyholder as the identity attributes are bound to the key. Legitimate users can recover the plaintext from the partially decrypted ciphertext using partial CP-ABE decryption.

In [15], Xiong et al. proposed a CP-ABE- based storage model for data storing and secure access in a cloud service for IoT applications. This scheme outsources decryption to the cloud server and dumps public key storage of both AAs and users to an attribute authority management module (AAM) which helps minimize computing and storage overhead in a system level.

Recently, Sanchol et al. [2] proposed a scheme to support data sharing in mobile cloud computing. This scheme outsources the full CP-ABE decryption process to the trusted proxy located in the cloud. The user only decrypts the ciphertext by using the symmetric key through the computation of the secret key which is computed upon the decryption. However, this work did not advocate for outsourced encryption.

In [16], the authors proposed a tag-aided encryption technique to protect item-level data protection for IoT records in a cloud-assisted industrial IoT environment. In this scheme, item keys are issued to participants for supporting secure communication while the IoT record is transferred.

In fact, the security data transmission and aggregation of health data collected from IoT devices are also crucial. Before the construction and storage of EMR data, raw data from various medical devices need to be collected first. There are a few works [17], [18] that focus on encrypting the IoT data before they are sent out from the devices. However, they do not provide fine-grained access control to encrypted IoT data.

Recently, several works [19], [20], [21], [22], [23], [24], [25], [26], [27], [28] engaged blockchain technology to collaborate with cloud computing to enable efficient data sharing and decentralized access control functions such as authentication, authorization, and accountability. For example, Liu et al. [21] proposed a blockchain-based privacy-preserving data sharing for EMRs. In this scheme, the EMRs are encrypted using CP-ABE before they are uploaded to the cloud. The ciphertext index is stored in an unalterable consortium blockchain. However, their process of CP-ABE decryption is not suitable for the IoT cloud environment.

In [19], the authors proposed the IoTChain model to provide fine-grained permission for IoT data. This scheme relies on attribute-based access control (A-BAC) and AES-128 encryption schemes to encrypt the IoT stream prior to uploading it to the IPFS. In addition, smart contracts are deployed on an Ethereum blockchain to implement encrypted keyword searches. However, the management of symmetric key management becomes the issue.

In [23], Zhang et al. proposed a blockchain-based hierarchical data sharing framework (BHDSF) to provide fine-grained access control and efficient retrieval over encrypted PHRs using CP-ABE and symmetric key encryption. The scheme employs online/offline and outsourced CP-ABE decryption techniques to offload most of the computation operations to the cloud server. In this scheme, the user needs to compute the symmetric key based on the ciphertext search and the intermediate ciphertext.

In [24], S. Fugkeaw et al. proposed a IoT blockchain-based access control for electronic medical health record based on outsourced encryption and decryption. Smart contracts were developed to provide user authentication checks.

In [25], Alshehri et al. proposed a dynamic secure access control using the blockchain (DSA-Block) model. In this scheme, the hyperelliptic curve cryptography (HECC) was used to provide the authentication of the IoT devices and users. The IoT records were secure using a differential privacy mechanism before they are stored in the cloud.

In [26], Cheikhrouhou et al. proposed a lightweight blockchain-based and fog-enabled remote patient monitoring system to provide secure and efficient access to patient-collected data. The proposed scheme aimed at reducing the delay in accessing data on the cloud by introducing the local chain to cooperate with the fog node to participate in the consensus protocol.

In [27], Ramadikha et al. proposed a privacy-preserving approach for secure misbehavior detection in lightweight IoMT devices. The authors applied privacy-preserving bidirectional long-short-term memory (BiLSTM) and augments the security in which the data is disguised through Ethereum smart contract environment.

In [28], Didone et al. introduced blockchain-based privacy enforcement that consists of the privacy enforcement layer and data release layer which requires the interaction of data owner privacy policy, and the enforcement of the policy is done through the consensus algorithm in the blockchain.

Nevertheless, all the above approaches did not work on data encryption inside IoT devices and provided secure IoT data aggregation. In addition, there have no works dedicated to support secure policy updates in the IoT cloud environment which is an essential feature for access control deployed in the evolvable data-sharing environment.

III. BACKGROUND

This section describes the basic concept of bilinear maps including the definition of access structure, access tree, and blockchain which is used as a part of cryptographic module in our proposed scheme.

A. BILINEAR MAPS

Let G_0 and G_1 be two multiplicative cyclic groups of prime order p. Let g be a generator of G_0 and e be a bilinear map, e: $G_0 \times G_0 \rightarrow G_1$. The bilinear map e has the following properties:

- Bilinearity: $\forall_{u,v} \in G_0$ and a, b$\in Z_p$, $e(u^a, v^b) = e(u, v)^{ab} = e(u^b, v^a)$
- NON-DEGENERACY: $e(g, g) \neq 1$
- COMPUTABILITY: $\forall_{u,v} \in G_0$, an EFFICIENTLY COMPUTATION of e(u,v) EXIST

Definition 1: Access Structure: Let a set $\{P_1, P_2, \ldots, P_n\}$ be given attribute. A collection $A \subset 2^{\{P_1, P_2, \ldots, P_n\}}$ is monotone if $\forall B, C$: $if B \in A and B \subset C \longrightarrow CA$. An access structure is respectively be a monotone collection A of non-empty subsets of $\{P_1, P_2, \ldots, P_n\}$, i.e. $A \subset 2^{\{P_1, P_2, \ldots, P_n\}}/\{\emptyset\}$.

Definition 2: Access Tree T: Let T be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children, and a threshold value. If num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate, and when $k_x = num_x$, it is an AND gate. Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$. If the *k-of-n* gate is allowed in T, in this case, $k_x = k$ where k is the threshold value determined in the *k-of-n* gate.

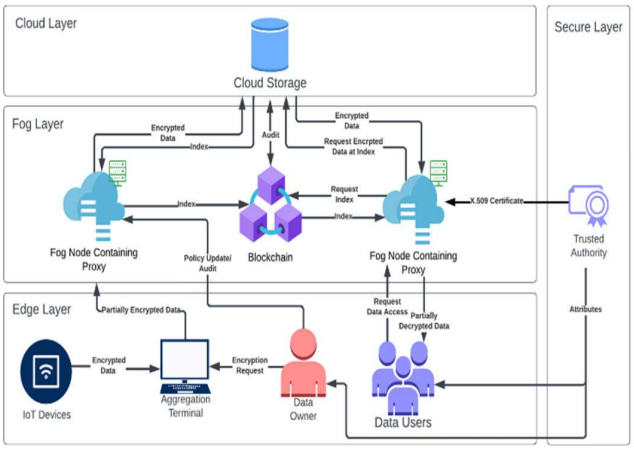

FIGURE 1. Our System Model.

B. BLOCKCHAIN

A Blockchain commonly generates a block of data by miners or validators on POW and POS, respectively. Proof of Work consensus requires a lot of computational power to solve complicated puzzles to determine the next block in the chain from particular parties. Proof of Stake consensus depends on users staking and the amount they are willing to lock up for a certain period of time. Each nonce in the blockchain is hashed by SHA-256 and currently there is no brute force attack to break it yet. In our proposed scheme, we used the Ethereum POS blockchain because it involves smart contracts to automatically run functions to control and manage data blocks. A smart contract is written through solidity as an executable to authenticate users, create index of the ciphertext, and record access transactions.

IV. OUR PROPOSED SCEHME

A. SYSTEM MODEL

Our system model comprises eight different types of entities:

Cloud Service Providers (CSPs), Fog Nodes (FNs), IoT Devices, Aggregation Terminal, Blockchain, Trusted Authority (TA), Data Owners (DOs), and Data Users (DUs). The treatment and measured data for each patient are collected via IoT devices and collected at an Aggregation terminal. Each aggregation terminal processes collected data and forwards them to the nearest fog node. Fog Nodes then complete the final process of computing and relaying data to the cloud storage and create indices of the ciphertext to store them on the blockchain. DU can access the data encrypted EMRs by making a request through their nearest fog node. The fog node then communicates with the blockchain to activate the smart contract for authentication and user identity verification. Upon the successful authentication, the index of the requested ciphertext is retrieved and the system then gains access to the requested ciphertext and encrypted random

value. The encrypted random value is then decrypted by the fog node before the derived random value is sent along with the ciphertext to the user. DU computes the symmetric key based on the random value and KeyGetFunction. After getting the symmetric key, the user can decrypt the ciphertext. Our proposed scheme's system model is shown in Figure 1.

The details of each system entity are described as follows.

1) Cloud Service Provider (CSP) provides data storage where EMRs are stored.
2) Fog Nodes house a set of computational resources to support computation tasks before the processed data are sent to store on the cloud. In our model, we offload both the encryption and decryption processes to be done in fog nodes. This helps reduce workload from edge devices that might lack computational power as well as reduce communication costs compared to using a proxy on the cloud. Furthermore, fog computing has the ability to communicate between adjacent nodes and can be used to iterate this model in future advancements using parallel computing. Due to fog nodes being semi-trusted entities, we apply a trusted proxy installed with a public key certificate to each fog node to make sure that there aren't any compromised fog nodes that could harm the system.
3) IoT Devices are devices or wearable sensors used to collect physiological data such as temperature, pressure rate, electrocardiograph (ECG), etc. from the patient's body.
4) Aggregation Terminal is employed to gather and transmit data from IoT devices. Before forwarding the data to the closest fog node, authorized personnel, such as a medical doctor, must decrypt and package the data. The device subsequently performs partial encryption on the data before transmitting it as partial ciphertext. IoT devices are employed to collect diverse patient data, such as blood pressure and oxygen levels, and can transmit this data over a network. In our approach, each device is equipped with an encryption algorithm that encrypts the data prior to its transmission to the aggregation terminal.
5) Blockchain includes a multitude of functions, including ciphertext index storage, auditing, session management, and authentication. Ciphertext Index storage is utilized to store and retrieve EMRs when fog nodes issue requests. Storing data on the Cloud service provider, the index is then stored on the Blockchain. Auditing is achieved due to the fact of immutable data records logged in the blockchain. In our scheme, three smart contracts are developed to perform (1) index storing, (2) user authentication and verification, and (3) session management. The indexing smart contract bundles the important parameters and index of encrypted ciphertext together for secure data retrieving and storing. The user authentication and verification smart contract checks the user identity and verifies the signature of the ciphertext indexing where the user requests to access. Finally, the session management contract is used to check the time and number of attempts for each user authentication session. All such misconduct activities are recorded in the system.

TABLE 1. Notations used.

Notation	Description
TA_k	The Trusted Authority k
D_{id}	An IoT device identified with a unique id
P_{id}	A patient with a unique id
U_{id}	A user with a unique id
S_E	A set of attributes issued to an Entity E
SK_E	A set of attributes issued to an Entity E and managed by TA_k
PK	Public attribute key issued by TA_k
MK	Master attribute key issued by the authority for SK_{DO} and SK_{DU} generation
RV	Random 256-bit String generated by a Cryptographically secure pseudorandom number generator
$KeyGetFN$	Function to generate a symmetric key
$SymKey$	AES key (256-bit) generated by $KeyGenFN$ and RV
N_P	The number of IoT devices used for P
M	A message or data shared in the cloud.
CT_E	The ciphertext of Entity E
$KeyRefid$	A reference value for fetching data user's SK_{DU}
F_{ad}	An aggregation terminal's secret function, $F_{ad} = \{f_{ad1}, f_{ad2}, .., f_{adNp}\}$
δp	A permuted list of a series of P
I_E	An index of Entity E
Sig	A value used for generating signature of bundle of data I_k in the blockchain
S_i	Secret Integer of ISC_{priv}
SV	Secret Integer of ISC_{pub}
P_{pubkp}	Public key Point
ISC_{priv}	The Private key of Indexing Smart Contracts
ISC_{pub}	The Public key of Indexing Smart Contracts

6) The Trusted Authority (TA) is responsible for generating, distributing, revoking, and managing the key and security parameters.
7) Data Owners (DOs) are individuals for whom data is collected by IoT devices. This includes patients as well as doctors who act on behalf of the patients.
8) Data Users (DUs) are medical professionals who require access to the stored data for treatment purposes. This includes doctors, nurses, physicians, or any other healthcare professionals.

B. CRYPTOGRAPHIC CONSTRUCT

This section presents our cryptographic protocols. Table 1. lists notations used in this paper.

The proposed method involves seven distinct stages, which include System Initialization, Key Generation, IoT Encryption, Data Aggregation, Data Encryption, Data Decryption, and Policy Updating.

1) PHASE1: SYSTEM INITIALIZATION

The System Initialization phase comprises three primary algorithms, namely initializeTrustedAuthority, which is executed by the TA, keyGetFuncGenerator, performed by each data owner, and dataAggFuncGenerator, carried out by each Aggregation Terminal.

1. initializeTrustedAuthority(κ) $\rightarrow$ (PK, MK)

The first algorithm in the System Initialization phase accepts a single input, which is a security parameter, and generates two outputs: the public key, PK and the master key, MK. This function selects a bilinear group of prime order and a generator, and then chooses two random values. The public key can be represented as follows:

$$PK_k = \{G_0, g, h = g^{\beta}, f = g^{\frac{1}{\beta}}, e(g,g)^{\alpha}\}$$

The master key, denoted as MK, can be expressed as follows:

$$MK_k = (\beta, g^{\alpha})$$

2. keyGetFuncGenerator()$\rightarrow$ ($KeyGetFN$)

This algorithm generates a KeyGetFN, which operates in the following manner:

```
keyGetFuncGenerator: Function {
SecureRandomKey secureRandKey =
CSPRNG.generateRandomKey()
return new keyGetFunc(string RV): String {
      For each RV as s {
            a = secureRandKey ⊕ s
      }
        return a
    }
}
```

The secureRandomKey value is generated using a cryptographically secure pseudorandom number generator (CSPRNG) and is a 256-bit value that is utilized for securely generating the keyGetFunc. Once each keyGetFunc has been created, they are transmitted to each DU within the system via a secure channel.

3. dataAggFuncGenerator()$\rightarrow$ (F_{ad})

This function generates a *Fad* for each aggregation terminal, which works as follows:

```
dataAggFuncGenerator: Function {
      SecureRandomVal secureRandVal =
      CSPRNG.generateRandomVal()
      return new adFunc(String RV,Int sr):String{
            Int start = 256 × sr
            secureRandVal =
            secureRandVal[start, start+256]
            For each RV as s {
                  a = secureRandVal ⊕ s
            }
            return a
      }
}
```

The secureRandomVal value is generated using a cryptographically secure pseudorandom number generator (CSPRNG), which is used for securely generating the adFunc, similar to the keyGetFuncGenerator algorithm. Once the adFunc has been created for each aggregation terminal, they are transmitted to the corresponding device within the system via a secure channel.

2) PHASE 2: KEY GENERATION

The key generation phase is responsible for creating and distributing keys and attributes to every DU and DO in the system. Moreover, a random number from a permutation series and partial function will also be given for each patient's IoT device. The TA executes three algorithms during this phase, namely duKeygen, doKeygen, and iotSetup.

1. duKeygen(PK, MK, S_{DU}) $\rightarrow$ (SK_{DU})

This algorithm takes as input PK, MK, and S_{DU}. The SK of the DU is generated by the duKeygen algorithm using a randomly selected value $r \in Z_p$, and each attribute $j \in S$ will be represented by randomly selecting $r_j \in Z_p$, resulting in the following:

$SK_{DU} = (D{=}g^{(\alpha+r)/\beta}, \forall j \in S : D_j = g^r.H(j)^{rj}, D'_j = g^{rj})$

After the SK_{DU} is generated for each DU in the system, the TA sends both SK_{DU} and the corresponding PK to the DU. The DU then forwards the SK_{DU} to the nearest fog node, where the proxy encrypts it using its public key and stores it. The fog node will then return a *KeyRefid* to the DU, which will be used for indexing the SK_{DU} during the decryption phase. Additionally, the fog node propagates the new key and *KeyRefid* to adjacent fog nodes, where the same process of encryption, storing, and propagation is done, with the difference being that the index for storing the SK_{DU} is given.

2. doKeygen(PK, MK, S_{DO}) $\rightarrow$ (SK_{DO})

This algorithm takes as inputs PK and MK, and S_{DO}. The same process used in the duKeygen function is applied here.

Once the SK_{DO} is computed, the TA sends the key to the corresponding DO.

3. iotSetup(PK, MK, F_{ad}, N_P, Did) $\rightarrow$ (f_{adi}, $\delta p(i)$)

This algorithm takes as inputs PK, MK, F_{ad}, being the aggregation terminal's secret function, N_P, and D_{id}. The functionality is described as

iotSetup(PK, MK, F_{ad}, N_P) : Function, Int {
Int secureRandI =
CSPRNG.pickAndDiscardRandomVal(N_P)
return new adFunc(string V): String {
return adFunc(V, secureRandI)
},
secureRandI
}

The Random Integer $\delta p(i)$ and the function f_{adi} are forwarded to the corresponding IoT Device via a secure channel.

3) PHASE 3: IOT ENCRYPTION

This stage involves the encryption of data collected from IoT devices of patient P, which is executed on each device. This phase is to ensure that the IoT data generated is secure while it is transmitted. The encryption function is defined as follows:

encryptIoT(m, i, $\delta p(i)$,N_P, t) $\rightarrow$ (CT_i, σCT_i)

This encryption algorithm takes the raw data m, random integer $\delta p(i)$, timestamp t, IoT identifier i, and N_P, and outputs a ciphertext of the raw data CT_i and Signature σCT_i. This algorithm works in the following manner:

encryptIoT(m, i, $\delta p(i)$,N_P, t): String, String {

$$String\ C = m \oplus fadi(t|\delta P(i))$$

For each N_P as j{
if ($j \neq \delta p(i)$){
$d = f_{adi}(t|j)$
}else {
d = C
}
CT = CT|d
}
return CT, Sign(CT)
}

The CT and σCT are then bundled and sent toward an Aggregation terminal.

4) PHASE 4: DATA AGGREGATION

This phase involves the collection and decryption of the collected patient data from various IoT devices. There are two algorithms: decryptIoT and aggregateData, which are all run on the aggregation terminal.

1. decryptIoT(CTs, σCTs, t, N_P) $\rightarrow$(M)

This algorithm takes inputs as a collection of Ciphertext Strings, collection of Signature Strings, timestamp t, and N_P and it outputs a Raw Data String. The algorithmic detail is presented as follows:

decryptIoT(CTs, σCTs, t, N_P): String{
For each CTs, σ CT as c, s {
if (Hash(c) $\neq$ s){
omit(c)
}else{

$$k = k \oplus c$$

}
}
OTPad = $Fad(t|N_P)$
return $k \oplus OTPad$
}

After each execution of the decryptIoT function, the aggregation terminal stores raw data with the time stamp of the IoT devices. After sufficient data has been collected, the authorized user can then add additional remarks to the data and bundle all raw data by using the aggregateData algorithm.

2. aggregateData(Ms, AD)$\rightarrow$ (D)

The algorithm concatenates an array of strings Ms with a string Ad as input to generate a final output string D, which is used for encryption.

aggregateData(Ms, AD): String{
For each Ms as m {

$$D = D|m$$

}
return $D|AD$
}

5) PHASE 5: DATA ENCRYPTION

This phase comprises four major functions including (1) doEnc, and (2) policyTreeEnc run on the Aggregation terminal, (3) fogEnc run on a fog node, (4) createIndex run on the Blockchain using an SC, and (5) indexStore run from the same fog node.

1. doEnc(M, $KeyGetFN$) $\rightarrow$ (CT_M,, RV, CT_T)

The function utilizes the AES symmetric key encryption algorithm to partially encrypt the data M. It generates a symmetric key using KeyGetFN and a randomly generated number. This function takes in the message M and Key Generation Function, $KeyGenFN$ and returns the Ciphertext of the Message, CT_M, the randomly generated number RV, and the Encrypted policy tree CT_T.

doEnc(M, $KeyGetFN$): String, String, Tree{
SecureRandomKey secureRandKey =
CSPRNG.generateRandomKey()
String sk = $KeyGetFN$(secureRandKey)
String CT = AES-Encrypt(M, $SymKey$)
PolicyTree CT_T =
policyTreeEnc($T.rootNode$, t, null)
return CT, secureRandKey, CT_T
}

DoEnc is also called policyTreeEnc, which is a recursive In-order Tree function.

2. PolicyTreeEnc($Node$, t, ic))$\rightarrow$ CT_T

This function encrypts each node of a Tree using ciphertext from the parent of the current node, ic, and the timestamp of

the root node initial function call, t, to output and return a Tree of encrypted Nodes CT_{T}.

PolicyTreeEnc($Node$, t, ic): Tree{

$$Node = Node \oplus t \oplus ic$$

For each Children of Node as c{
policyTreeEnc(c, t, Node)
}
}

Once the doEnc and policyTreeEnc algorithms have been executed, the resulting outputs, along with the DO's public key, will be sent to the closest fog node.

3. fogEnc(RV, CT_T, PK) $\rightarrow$ (CT_{RV1},RV2)

The purpose of the fogEnc function is to encrypt the RV for secure storage in the blockchain using CP-ABE. This is done by taking the PK, RV, and CT_{T} as input arguments, and then performing the encryption process.

fogEnc(RV, CT_{T}, PK): String, String{
PolicyTree T =
policyTreeDec($CT_{\mathrm{T}}.rootNode$, t, null)
$RV1, RV2$ =$RV.split()$
return CP-ABE-Encrypt($RV1$, PK, T), $RV2$
}

From a cryptographic perspective, the encryption process involves encrypting a random value $RV1$ under the access structure T. This is done by selecting a polynomial q_x for each node x in the tree T, including the leaves, in a top-down approach starting from the root node R. The degree dx of the polynomial q_x is set to be one less than the threshold value k_x of the node x. A random value s is chosen for q_x (0), and d_R other points are chosen randomly to define it completely. For any other node x, $q_{parent(x)}(index(x))$ is set as the value of $q_x(0)$ and d_x other points are chosen randomly to define q_x completely. Finally, the ciphertext is constructed using the tree access structure T and the above-defined polynomials.

$$CT_{RV1} = (T, C = RV1e(g,g)^{\alpha s},$$
$$C = h^s, \forall_y \in Y :$$
$$C_y = g^{q_y(0)}, C\prime_y = H(att(y))^{q_y(0)}).$$

fogEnc calls another function called policyTreeDec, which is used to decrypt the Policy Tree, so the CP-ABE encryption process can be completed. The algorithm can be seen as follows:

policyTreeDec($Node$, t, ic): Tree{
For each Child of Node as c{
policyTreeEnc(c, t, $Node$)
}

$$Node = Node \oplus t \oplus ic$$

}

After CT_{RV1} and $RV2$ arrive, the fog node then forwards $\mathrm{CT_M}$ and CT_{RV1} to the cloud to store the ciphertext. To get the ciphertext of $\mathrm{CT_M}$ and CT_{RV1} from the cloud, createIndex function is executed as below.

4. createIndex()$\rightarrow$ (I_{CT_M}, $I_{CT_{M,RV}}$)

This function creates indices for the Ciphertext CT_{M} and CT_{RV1} by extracting the storage location of the latter and bundling them as separate indices. This function is run by fog nodes with the presence of the cloud.

createIndex():String, String {
$\mathbf{I}_{\boldsymbol{CT_M}}$ = transaction.getLocation(CT_{M})
$I_{CT_{M,RV}}$ = transaction.getLocation(CT_{RV1})
return $\mathbf{I}_{\boldsymbol{CT_M}}$, $I_{CT_{M,RV}}$
}

After receiving both indexes of the Ciphertext CT_{M} and CT_{RV1} as I_{CT_M}, $\mathrm{I_{CT_{M,RV}}}$ from the cloud, fog node will forward them to blockchain. Then, SC will bundle the I_{CT_M} and $I_{CT_{M,RV}}$ with $RV2$ from the Fog Node and store into the blockchain using the following function of indexStore.

5. indexStore($RV2$, I_{CT_M}, $I_{CT_{M,RV}}$)

This function stores the RV2, I_{CT_M}, and $I_{CT_{M,RV}}$ as a bundle onto the blockchain from the Fog node. In [20], Saini et al. proposed a way of generating the signature for securing authentication of the EMR. In our scheme, we used the EdDSA algorithm to generate signature. We take advantage of the fact that the curve comes with a subgroup order q that is generated from G. By using this concept, we can avoid having to perform additional computations during the key generation process. We only used RV2 as the public indexing smart contract $\mathrm{ISC_{pub}}$ and the XOR operation between I_{CT_M} and $I_{CT_{M,RV}}$ as the private indexing smart contract ISC_{priv}. For the purpose of signing the indexing smart contract, we require the public key point P_{pubk} and a value for generated signature Sig. The key pair of the indexing smart contract $\mathrm{ISC_{pub}}$/$\mathrm{ISC_{priv}}$ and indexStore function are defined as follows:

indexStore($RV2$, I_{CT_M}, $I_{CT_{M,RV}}$): {
$\mathrm{ISC_{pub}} = \mathrm{RV2}$
$\mathrm{ISC_{priv}} = I_{CT_M} \oplus I_{CT_{M,RV}}$
$I(I_{CT_M}, I_{CT_{M,RV}}, RV2) = \mathrm{tuple}(RV2, I_{CT_M}, I_{CT_{M,RV}})$
$\mathrm{I_K} = I(I_{CT_M}, I_{CT_{M,RV}}, RV2)$
S_{i} =Hash(Hash(ISC_{priv}) +I_K) mod q
$P_{pubkp} = S_i * G$
SV= Hash(P_{pubkp}+ ISC_{pub} + I_K) mod q
Sig= (S_i + SV + ISC_{priv}) mod q
(I_K, ISC_{priv}) -> {P_{pubkp}, Sig} = ISC_{sign}
}

The signature is generated by taking as input both indices of the ciphertext of encrypted data EMRs, the ciphertext of RV1 used in the CP-ABE algorithm, and RV2. The overall indexStore() function does only basic operations such as xor, mod, and hashing repeatedly to get $\mathrm{ISC_{sign}}$ for later use in the verification process of the second SC.

6) PHASE 6: DATA DECRYPTION

In this phase, patient data stored on the cloud is requested and decrypted. This phase comprises three algorithms: authenticateDU, which is run by the blockchain via SC; decryptFog, which is executed on the fog node; and decryptDU, which

is performed by the requesting DU. The process begins with the DU making a request to the nearest fog node, providing the KeyRefid of the data user for indexing the SK_{DU} in the fog node's storage. For the decryption process, the authentication and verification smart contract firstly validates the user by taking inputs such as UserID, a bundle of encrypted files ($\mathrm{I_K}$), and the public key of indexing smart contract ($\mathrm{ISC_{priv}}$). In this stage, the decryption request is sent to the fog node to activate authenticateDU() function and perform the decryption as described by the algorithms as follows.

1. authenticateDU()→(AccessStatus, $RV2$, I_K)

This function authenticates the user if the request is legitimate by the Blockchain.

```
authenticateDU():{
  registration(PIIi) if true{
    validation(IDi , IK, ISCpub) if true{
      requestCheck()
      signature_Verification() if true{
        grantAcess() if true{
          misConduct(True, Status)
        }revoke()
      }revoke()
    }revoke()
  }
}
```

Based on the above algorithm, if the data user is granted with the access to the requested data, the Smart Contract (SC) returns the indices of the encrypted Data ($\mathrm{CT_M}$), encrypted Random Value 1 ($\mathrm{CT_{RV1}}$), and Random Value 2 (RV2). Then, the fog node sends a request to the cloud to retrieve the data stored at the indices I_{CT_M} and $I_{CT_{M,RV1}}$.

2. decryptFog($\mathrm{CT_{RV1}}$, SK_{DU}) → ($RV1$)

The decryptFog function takes the encrypted $RV1$ and a the SK_{DU} as inputs and it outputs the $RV1$. The decryption process involves retrieving the KeyRefid of the SK_{DU} from the Fog node's storage using the index provided in the input. Then the SK_{DU} is used to decrypt the $\mathrm{CT_{RV1}}$, and the $RV1$ is returned.

```
decryptFog(CTRV1, SKDU): String {
return CP-ABE-Decrypt(CTRV1, SKDU)
}
```

After the successful decryption of $RV1$, the fog node retrieves the encrypted $\mathrm{CT_M}$ from the cloud using their respective index and $RV2$ from the Blockchain. These values, along with the $RV1$, are then sent back to the DU who made the request.

3. decryptDU($KeyGetFN$, $RV1$, $RV2$, $\mathrm{CT_M}$) → (M)

The user runs decryptDU to decrypt the ciphertext by using the $KeyGenFN$, $RV1$, $andRV2$ to obtain the symmetric key. The decryption function is defined as:

```
decryptDU(KeyGetFN, RV1, RV2, CTM): String {
String sk = KeyGetFN(RV1, RV2)
return AES-Decrypt(CTM, sk)
}
```

Finally, M is then obtained.

Upon the decryption, the data users do not need to retain the derived symmetric key in their terminal or device. The freshest key can be computed when the decryption is needed.

7) PHASE 7: POLICY UPDATING

This stage contains the main functionality of updating the policy of encrypted EMRs on the Cloud. This phase consists of two functions, initPolicyUpdate, and updatePolicy, which are executed on the fog node.

1. initPolicyUpdate(T) → (CT_{T})

This function encrypts the new Policy Tree, T, using the same function as in the Data Encryption phase.

```
initPolicyUpdate(T): String {
return policyTreeEnc(T.rootNode, t, null)
}
```

After the CT_{T} created, the DO sends a policy update Request with the CT_{T} to the nearest fog node. Additionally, the $KeyRefid$ of the DO used by the fog node for creating index of the SK_{DU} is also forwarded.

2. updatePolicy(CT_{T}, CT_{RV1}, SK_{DU}) → (CT'_{RV1})

This function decrypts the encrypted Policy Tree CT_{T}, the cipher text of $RV1$, CT_{RV1} using the secret key SK_{DU}, and returns the new cipher text of RV1, CT'_{RV1}.

```
updatePolicy(CTT, CTRV1, SKDU): String {
PolicyTree T =
policyTreeDec(CTT.rootNode, t, null)
String RV1 = CP-ABE_Decrypt(CTRV1, SKDU)
return CP-ABE_Encrypt(RV1, PK, T)
}
```

The method policyTreeDec from the Data Encryption phase is called here to retrieve the usable form of the Policy Tree T. Afterwards, this policy tree is used to encrypt RV1, which is then stored on the cloud. Hence, the computing tasks related to the policy update are done in the fog node. This helps reduce both latency for processing the requests and the computation burden to be done by the data owners.

V. SECURITY ANALYSIS

This section provides an analysis of the security of our proposed scheme including security model and some security properties.

A. SECURITY MODEL

In this paper, the security is proven in a game-based manner. The main security aspect that is used for storing data is CP-ABE, which is further elaborated in the reference to the CP-ABE scheme's original publication [1].

In the scheme's environment, where the data is stored in a cloud environment, and the secret key for decryption is delegated to the fog nodes. DOs are considered to be fully trusted entities, whereas DUs are assumed to be not fully trusted. Additionally, it is assumed that key queries are made in an adaptive manner, while Adversary A has the ability to corrupt the Trusted Authority statically. In our proposed

scheme, the security of CP-ABE is analyzed through the interaction between an Adversary A and a Challenger C.

Theorem 1: Suppose no adversary in polynomial time who can break the security aspect of CP-ABE with non-negligible advantage exists, then there is no adversary in polynomial time who can break the security of our system with non-negligible advantage.

1) Initialization

A key pair is created by a TA and is forwarded to the simulator, which also generates an additional key pair. The Challenger C then runs initializeTrustedAuthority for all the creditable authorities in SA - $S'A$ and afterward sends a public key PK to Adversary A. However, the challenger forwards both the public key and secret key to the Adversary A for all corrupted Authorities $S'A$.

Phase I

The Adversary A sends the set of attributes S from TA to the Challenger C, which belongs to the uncorrupted authority. Finally, the Challenger gives the secret key SK that corresponds to the attributes that are part of S to the Adversary A.

2) Challenge

The Adversary A sends two messages $m0$ and $m1$, which are in equal length to the simulator. A fair binary coin β is then flipped by the simulator, where encryption of $m\beta$ is returned. The encrypted $m\beta$ is computed as follows:

$CT = (T, C" = m\beta Z, CT = hS, \forall y \in Y : Cy = g^{qy}(0),$

$C'y = H(att(y))qy(0))$ where y is a chosen set of attributes.

If $\mu = 0$ then $z = e(g, g)\alpha s$. Hence, it can be said that the ciphertext is a valid random encryption of $m\beta$.

Else if $\mu = 1$ then $z = e(g, g)z$. Which will then yield $C" = m\beta e(g, g)^z$. Because z is considered random, C will also be a random element of G_1 from the perspective of the Adversary A, hence the message contains no information about $m\beta$.

Phase II

The same process is performed by the simulator as in *Phase I*.

A Guess is made by the Adversary A, as β' of β. By this Adversary A's advantage is computed as follows:

Advantage$= Pr[\beta = \beta'] - 1/2$

Our scheme is thereby secure if all Adversaries in polynomial time have at most a negligible advantage in the shown game above.

Theorem 2: An Adversary cannot decrypt a ciphertext file using only the partial value decrypted from a corrupted fog node.

3) Decryption

Suppose an Adversary A gets hold of the partial random value that is encrypted using CP-ABE. In order to get hold of the actual data that is encrypted in the AES format, the Adversary A would additionally need the Secret Key generation function of the Data Owner $KeyGetFN$, and the second partial random value in order to regenerate the AES symmetric key. This is due to the fact that the second part of the random value is stored on the Blockchain, where smart contracts handle authentication and auditing.

Theorem 3: Suppose no adversary in polynomial time who can distinguish δp and its random permutation $\delta' p$, υP cannot be computationally distinguished from $\upsilon' P$.

4) IoT Encryption

Assuming an Adversary A gets hold of transmitted ciphertexts from IoT Devices. As each device receives its random number from the permutated list $\delta' p$ from δp, the Adversary A must know what the value of the given permutation belongs to the device in order to derive the position of the IoT device's sent data, as the ciphertext string contains dummy data concatenated. Furthermore, the Aggregation terminal's secret function is needed to obtain the value that is needed to decrypt the data in the first place, which once again requires the random permutation $\delta' p$. Therefore, in order to get the actual value υP from all other $\upsilon' P$,the Adversary A has to first distinguish what value $\delta' p$ has from δp.

B. CONFIDENTIALLY OF POLICIES

In our scheme, we offloaded CP-ABE encryption and policy updating to fog nodes, where the policy tree used in these processes is encrypted. All attributes in the policy tree are encrypted in a chained manner, using an In-order algorithm and the timestamp that is selected by the DO. The uniqueness of each timestamp used, and the usage of parent encryption embedded in child nodes, leads to untraceable and undistinguishable ciphertext for each node, even if the same attributes occur multiple times.

C. IOT ENCRYPTION SECURITY

The method policyTreeDec from the Data Encryption phase is called to retrieve the usable form of the Policy Tree T. Afterwards, this Policy Tree is used to encrypt RV1, which is then stored on the cloud.

VI. EVALUATION

In this section, we present a comparative analysis of our LightMED scheme and related works based on the functionality and computation cost. Specifically, we consider works using CP-ABE supporting secure and fine-grained sharing of IoT data. We also provide details of experiments regarding the performance test of our scheme and the related works.

A. FUNCTIONALITY COMPARISON

Table 2 presents a comparison of the functionalities of our proposed scheme and related works including [3], and [23].

In summary, all the compared schemes in this study support IoT data encryption, but with some differences in functionality and computation. While schemes [3] and [23] encrypt the data after it is generated from the device, our proposed scheme encrypts the IoT data inside the device before aggregation. In terms of authentication of IoT

TABLE 2. Functionality comparison.

Scheme	F1	F2	F3	F4	F5	F6
[3]	x	x	x	x	✓	x
[23]	x	✓	✓	x	x	✓
Ours	✓	✓	✓	✓	✓	✓

Note: F1 = IoT Data encryption by IoT device; F2=Authentication of IoT data source; F3 = Blockchain-assisted; F4 = Outsourced encryption; F5 = Outsourced decryption; F6 = Policy update

data source, scheme [23] uses aggregative authentication, while our scheme uses the ECDSA algorithm. Regarding the cryptographic outsourcing model, our proposed scheme provides full CP-ABE encryption and decryption to be done in the fog node, whereas schemes [3] and [23] outsource partial CP-ABE decryption to be done by the cloud. Lastly, our scheme includes a policy update function that enables the data owner to update the access policy with optimized re-encryption cost performed by the fog node.

B. COMPUTATION COST ANALYSIS

Table 3 presents the comparison of the computation cost of our scheme and related works. In order to describe the representation of computation cost of each scheme, the following notations are used.

$|A_O|$: The number of attributes owned by data owner.
$|A_U|$: The number of attributes owned by data user.
$|T_A|$: The number of leaf nodes in access control policy.
E_{G0}, E_{G1} : The size of element in G_0 and G_1 groups.
G_0: Exponentiation and XOR operation in group G_0
G_1: Exponentiation and XOR operation in group G_1
G_e: Pairing operation in group G_0
G_m: Multiplication operation in group G_0
Z_p : The group $\{0,1,\ldots,p\text{-}1\}$ multiplication modulo p
XOR: XOR operation in 256bits data.
E_{KeyGFn}: The size of element in $KeyG_{FN}$. 256bytes
AES: AES encryption/decryption operation.

From Table 3, the computation cost of encryption of scheme [3] is subject to full CP-ABE and symmetric encryption at the data owner side. The encryption time increases in proportion to the number of attributes in the access policy. In [23], the encryption cost consists of offline encryption and online encryption based on the pairing and exponentiation over the LSSS structure. In addition, the computation cost of this scheme is subject to the number of keywords used to search in the ciphertext. In our scheme, there is an encryption cost done at the IoT device. At the data owner side, there is a cost of symmetric encryption while the CP-ABE encryption is offloaded to the fog node. For the decryption cost, our scheme gives least cost for data user as the user only deals with the AES decryption while scheme [3] and [23] the users need to perform partial CP-ABE decryption.

TABLE 3. Comparison of computation cost.

Scheme	Encryption Cost		Decryption Cost	
	Data Owner	Provider or Proxy	Data User	Provider or Proxy
[3]	*AES*+ $(\|T_A\|+1)G_0$ + $2G_1$	N/A	*AES*+ $(\|T_A\|+2)G_1$	$\|A_U\|G_e + \|T_A\|G_1$
[23]	$(\|T_A\|+4)G_0$ +G_1	$\|T_A\|G_0$	$3G_0 + G_1$	$(3\|A_u\|+5)G_e + G_0$ + $(\|T_A\| + 1)\ G_1$
Ours	*AES* + *XOR*	$(2\|T_A\|+1)$ $G_0 + 2G_1$	*AES* + *XOR*	$(2\|A_u\|+1)G_e$ + $(2\|T_A\|+2)G_1$

C. PERFORMANCE ANALYSIS

In this section, we did the simulations through the experiments to evaluate encryption, decryption, and policy update performance. The steps of our experiments are done with setting up the environment for IoT data generation and encryption, a proxy for data encryption and decryption, and the blockchain. Afterward, we implemented our logic programmatically using Python for the Cryptographic Operations, and Solidity for the Smart Contracts Proof of Concept. Finally, we used timers that each programming language offers to measure the performance of each operation in the unit of nanoseconds, which we then convert to either milliseconds or seconds as seems fit.

1) EXPERIMENT ENVIRONMENT SETUP

To evaluate the performance of our proposed scheme, we did experiments to measure the encryption and decryption cost of our proposed scheme and the systems in [3], and [23]. The implementation is done via Python's Cryptography and we used Java-Pairing based Cryptography [29] and the Advanced Crypto Software Collection [30], [31] to simulate the cryptographic operations of all schemes. The experiments were done on an Intel(R) Xeon(R) E-2236 CPU @ 3.40GHz and 16 GB of RAM server that is running on the Ubuntu 20.04 Operating System. When it comes to measuring the throughput of our system, we took advantage of Python's multithreading library to simulate having multiple concurrent processes running simultaneously. The parameters used in our experiments are shown in Table 4 .

2) BLOCKCHAIN ENVIRONMENT SETUP

We implemented smart contracts on the Ethereum network [32] using the Solidity programming language with the Web-based developer environment of Remix.Ethereum online IDE. This IDE supports the developing, deploying, debugging, and testing of Ethereum smart contracts. We only test it on the local environment as the default workspace for

TABLE 4. Experimentation variables and parameters.

Parameter	Setting
Data Size	40 KB
AES Key length	256 Bits
RV length	256 Bits
Number of Policies (Encryption & Decryption)	1, 2, 5, 10, 20, 40, 80, 100
Number of Policies (Policy Update)	20
Number of Concurrent Policy Update Requests	1, 2, 4, 10, 50, 100, 200, 500, 1000, 1500, 3000, 5000, 10000, 25000, 50000, 100000

testing purposes without the cross-site wallet or any particular Ethereum addresses. For the solidity IDE, we used the compiler version 0.8.18+ commit and the environment of Remix VM (Shanghai) as the virtual machine. In our system, a smart contract randomly generates 100 user IDs from their personal information with inputString values representing the ISC_{pub} and assignedString representing the index of a bundle of ciphertext. The signature is randomly generated with 128-bit strings and only the valid signature can be used as the additional authentication mechanism to allow authorized users to access the ciphertext.

We also provided the source code of our proposed core cryptographic functions and Smart Contracts in Github as in [33].

3) ENCRYPTION AND DECRYPTION PERFORMANCE

In the conducted experiments, the processes of Data owner Encryption, Proxy Encryption, Data user Decryption, and Proxy Decryption were isolated. We used parameters such as data size, AES Key length, and the number of attributes in the Access Tree to ensure accurate comparisons between the tested systems. The computation time was then measured by varying the number of attributes in the access policy while maintaining a constant data amount of 40 KB for all executions of all implemented algorithms, which includes ours, Hahn et al.'s scheme [3], and Zhang et al.'s scheme [23]. Figure 2 and Figure 3 provide a breakdown of the encryption and decryption costs incurred by various entities, including the Data Owner, Proxy/Fog node, and Data User, for all schemes. Furthermore, Figure 4 and Figure 5 depict the overall encryption and decryption costs for all schemes.

Figure 2 shows the comparison in computational time of each scheme compared to ours, where each location of cryptographic operations is separated. This shows that our scheme's encryption times at the DO side stay constant and minimal at all variations of the attribute numbers, compared to the other schemes. Figure 3 shows that the computational time needed for decryption at the DU side is constant and consistently lower compared to all the other schemes, at all numbers of attributes. This is due to the fact that our proposed

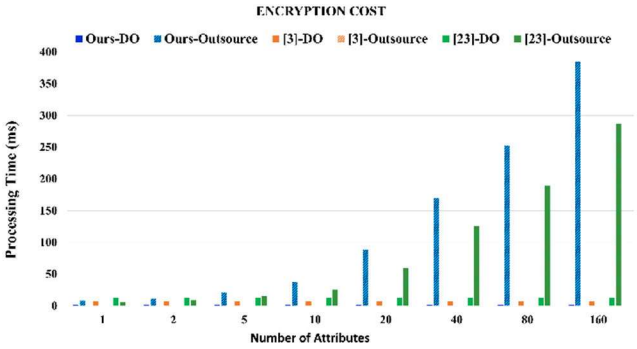

FIGURE 2. Encryption cost that incurs at each entity.

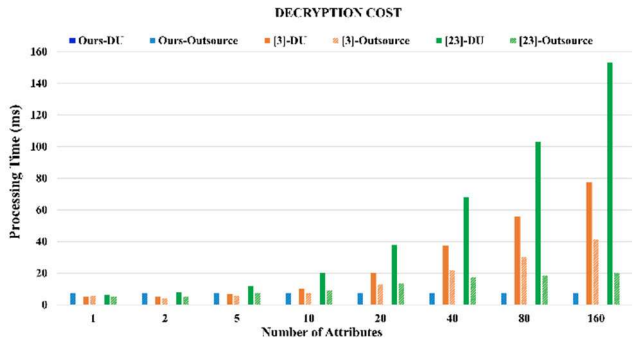

FIGURE 3. Decryption cost that incurs at each entity.

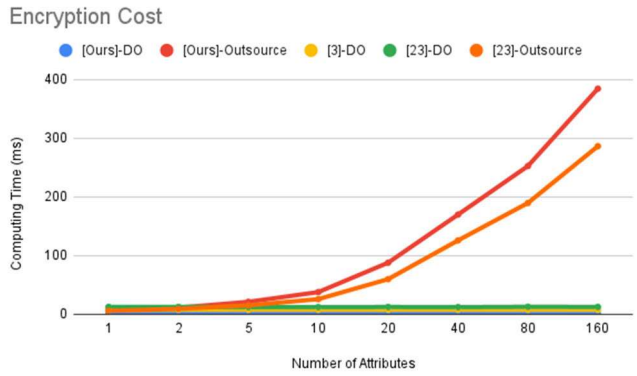

FIGURE 4. Total encryption cost.

scheme only requires the DO and DU to perform AES encryption and decryption, which is comparatively cheaper to the other schemes.

Figure. 4 shows the total encryption cost of the three schemes, while Figure 5 represents the total decryption cost. For encryption, our scheme yields a constant computational time and has the least cost as it relies solely on AES encryption while schemes [3] and [23] provide a bit higher cost at the DO side as they deal with partial CP-ABE encryption that scales with the number of attributes in the access structure.

However, our scheme gives the highest cost at the outsourcing node, which is negligible due to the fact that we

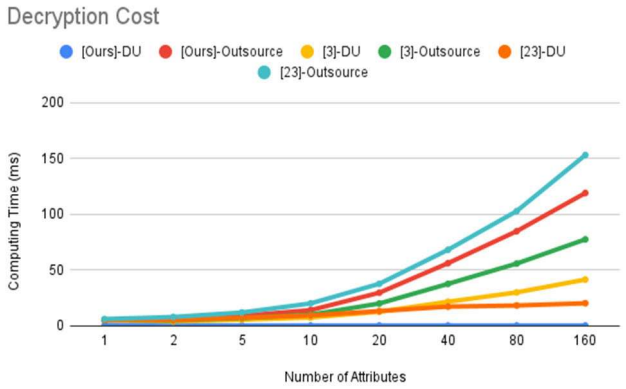

FIGURE 5. Total decryption cost.

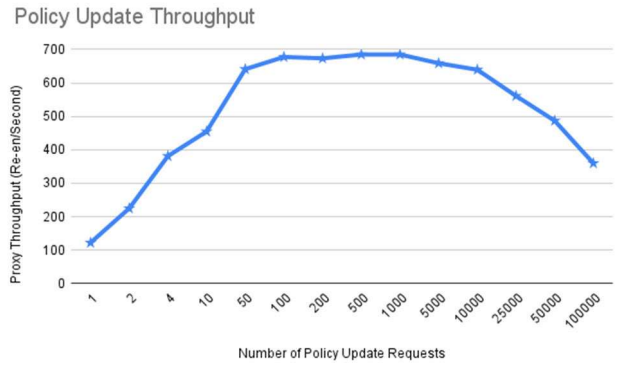

FIGURE 6. Throughput of policy updating.

impose the assumption of outsourced nodes have a scalable and cheap infrastructure. For decryption, Figure 5 shows that in schemes [3] and [23], the decryption time increases upon the increased number of attributes in the access structure. Similarly, the cost of decryption at the outsourcing node is subject to the number of attributes as there was a full CP-ABE decryption. However, our scheme yielded constant and least decryption cost at the DU side compared to [3] and [23].

In summary, the experimental results confirmed that our scheme renders the least cost for both encryption and decryption costs done at both the DO and DU sides.

4) POLICY UPDATE THROUGHPUT

To evaluate the efficiency of our proposed policy update algorithm, we conducted the test to measure the re-encryption cost done by the fog node when the policy is updated. The test aimed to determine how much our scheme accommodates the update transactions. The throughput has been measured using the generation of concurrent multi-thread requests. Here, we used the policy containing 5 attributes for the test.

From Figure 6, the result showed that our proposed scheme allows for a maximum of approximately 680 Re-encryptions per second from 100 to 1,000 concurrent policy update requests. However, the system can still handle a load that is higher in an optimistic manner since the biggest testing data is 100,000 concurrent requests, but the system has pulled through and had a high Re-Encryption per second rate of 359.71. This is bound to the aforementioned hardware specifications that we used and hence can be scaled up even more. Since the re-encryption process is done over a small size of random value, our scheme is practical to be implemented in the environment where a large number of users access the system.

VII. DISCUSSION

In essence, all experimental results indicate the applicability and practicality of the proposed scheme as they can represent the pattern of how the scheme's performance interacts with different load sizes. Specifically, our proposed scheme improves the efficiency of the encryption and decryption for end-users (data user/ data owner) since both encryption and decryption on end-users do not scale with the number of attributes. Furthermore, conducting policy updates is another critical operation that has a high possibility of being performed by several end-users at the same or different points in time. Regardless, for this experimentation, we can see a positive pattern of throughput referencing a higher number of re-encryption operations, which usually require higher computational power, showcasing that even with a high number of policy update requests our scheme can still handle the load.

From the design of our scheme, all of the encryption and decryption operations regarding CP-ABE are conducted on the fog layer, therefore the end-user's performance concern is tackled by this design choice. The Fog Layer is simulated by a single machine in our test environment, which does not represent the full potential of Fog Computing, as a major advantage of Fog Computing, sharing and adjusting resources from/to adjacent Fog Nodes. Utilizing full capacity of Fog computing could be a real challenge for our future work.

VIII. CONCLUSION

We have proposed the LightMED scheme to provide secure, fine-grained, and lightweight access control for outsourced IoT-EMRs using fog computing and blockchain. In our scheme, we proposed IoT data encryption and secure aggregation. To render fine-grained and lightweight data access, we fully outsource both CP-ABE encryption with privacy-preserving policy and decryption to the fog nodes that help reduce the overall communication and computation costs for both data owner and end-user. We employed blockchain to support decentralized authentication, data indexing, secret random parameters retention, and auditing. Furthermore, we proposed a lightweight policy update algorithm to support policy evolution in the IoT cloud environment. Finally, we conducted the experiments and the results showed that the performance of our proposed cryptographic operations outperforms the related works significantly and our system's throughput indicated that our scheme is efficient and highly scalable for real deployment. Nevertheless, complications

regarding user and attribute revocation have not been fully tackled and are one of the crucial priorities that we will be further improving upon in future developments and research concerning this field. In addition, value-guessing attacks on attributes are an outstanding issue as well. This problem has been partially tackled via our novel Policy Tree Encryption scheme. However, this only partially addresses the vulnerability as it does not cover the visibility of attributes in the ciphertext itself. With the base that we have built, we are positive and eager to continue improving the scheme to cover attributes hiding in our future work. Finally, applying the full utilization of Fog Computing can enhance our system's capabilities. This is due to the concept of intelligently sharing resources and dynamically offloading the same or similar operations with the same parameters to nearby Fog Nodes. This concept will significantly improve our system performance.

REFERENCES

[1] U. C. Yadav and S. T. Ali, "Ciphertext policy-hiding attribute-based encryption," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Aug. 2015, pp. 2067–2071, doi: 10.1109/ICACCI.2015.7275921.

[2] P. Sanchol, S. Fugkeaw, and H. Sato, "A mobile cloud-based access control with efficiently outsourced decryption," in *Proc. 10th IEEE Int. Conf. Mobile Cloud Comput., Services, Eng. (MobileCloud)*, Aug. 2022, pp. 1–8, doi: 10.1109/MobileCloud55333.2022.00008.

[3] C. Hahn, J. Kim, H. Kwon, and J. Hur, "Efficient IoT management with resilience to unauthorized access to cloud storage," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1008–1020, Apr. 2022, doi: 10.1109/TCC.2020.2985046.

[4] J. Lai, R. H. Deng, C. Guan, and J. Weng, "Attribute-based encryption with verifiable outsourced decryption," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 8, pp. 1343–1354, Aug. 2013.

[5] S. Lin, R. Zhang, H. Ma, and M. Wang, "Revisiting attribute-based encryption with verifiable outsourced decryption," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 10, pp. 2119–2130, Oct. 2015.

[6] S. Abdollahi, J. Mohajeri, and M. Salmasizadeh, "Highly efficient and revocable CP-ABE with outsourcing decryption for IoT," in *Proc. 18th Int. ISC Conf. Inf. Secur. Cryptol. (ISCISC)*, Sep. 2021, pp. 81–88, doi: 10.1109/ISCISC53448.2021.9720469.

[7] J. Li, Q. Huang, X. Chen, S. S. M. Chow, D. S. Wong, and D. Xie, "Multi-authority ciphertext-policy attribute-based encryption with accountability," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur.*, Mar. 2011, pp. 386–390.

[8] Z. Liu, Z. Cao, and D. S. Wong, "Blackbox traceable CP-ABE: How to catch people leaking their keys by selling decryption devices on ebay," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 475–486.

[9] Z. Liu and D. S. Wong, "Traceable CP-ABE on prime order groups: Fully secure and fully collusion-resistant blackbox traceable," in *Information and Communications Security* (Lecture Notes in Computer Science), vol. 9543, S. Qing, E. Okamoto, K. Kim, and D. Liu, Eds. Cham, Switzerland: Springer, 2016, pp. 109–124.

[10] S. Fugkeaw, "A lightweight policy update scheme for outsourced personal health records sharing," *IEEE Access*, vol. 9, pp. 54862–54871, 2021, doi: 10.1109/ACCESS.2021.3071150.

[11] W. Yuan, "Dynamic policy update for ciphertext-policy attribute-based encryption," *IACR Cryptol., ePrint Arch.* vol. 2016, pp. 457–468, May 2016.

[12] K. Yang, X. Jia, K. Ren, R. Xie, and L. Huang, "Enabling efficient access control with dynamic policy updating for big data in the cloud," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2014, pp. 2013–2021.

[13] J. Li, S. Wang, Y. Li, H. Wang, H. Wang, H. Wang, J. Chen, and Z. You, "An efficient attribute-based encryption scheme with policy update and file update in cloud computing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 12, pp. 6500–6509, Dec. 2019.

[14] J. Yu, S. Liu, S. Wang, Y. Xiao, and B. Yan, "LH-ABSC: A lightweight hybrid attribute-based signcryption scheme for cloud-fog-assisted IoT," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 7949–7966, Sep. 2020, doi: 10.1109/JIOT.2020.2992288.

[15] S. Xiong, Q. Ni, L. Wang, and Q. Wang, "SEM-ACSIT: Secure and efficient multiauthority access control for IoT cloud storage," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2914–2927, Apr. 2020, doi: 10.1109/JIOT.2020.2963899.

[16] S. Qi, Y. Lu, W. Wei, and X. Chen, "Efficient data access control with fine-grained data protection in cloud-assisted IIoT," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2886–2899, Feb. 2021, doi: 10.1109/JIOT.2020.3020979.

[17] H. Tao, M. Z. A. Bhuiyan, A. N. Abdalla, M. M. Hassan, J. M. Zain, and T. Hayajneh, "Secured data collection with hardware-based ciphers for IoT-based healthcare," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 410–420, Feb. 2019, doi: 10.1109/JIOT.2018.2854714.

[18] X. Shen, L. Zhu, C. Xu, K. Sharif, and R. Lu, "A privacy-preserving data aggregation scheme for dynamic groups in fog computing," *Inf. Sci.*, vol. 514, pp. 118–130, Apr. 2020, doi: 10.1016/j.ins.2019.12.007.

[19] Z. Ullah, B. Raza, H. Shah, S. Khan, and A. Waheed, "Towards blockchain-based secure storage and trusted data sharing scheme for IoT environment," *IEEE Access*, vol. 10, pp. 36978–36994, 2022, doi: 10.1109/ACCESS.2022.3164081.

[20] A. Saini, Q. Zhu, N. Singh, Y. Xiang, L. Gao, and Y. Zhang, "A smart-contract-based access control framework for cloud smart healthcare system," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5914–5925, Apr. 2021, doi: 10.1109/JIOT.2020.3032997.

[21] 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, doi: 10.1109/GLOCOM.2018.8647713.

[22] A. AlMamun, F. Jahangir, M. Umor, S. Azam, M. S. Kaiser, and A. Karim, "A combined framework of interplanetary file system and blockchain to securely manage electronic medical records," in *Proc. Int. Conf. Trends Comput. Cogn. Eng.* Singapore: Springer, 2021, pp. 501–511.

[23] J. Zhang, Y. Yang, X. Liu, and J. Ma, "An efficient blockchain-based hierarchical data sharing for healthcare Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 7139–7150, Oct. 2022, doi: 10.1109/TII.2022.3145851.

[24] S. Fugkeaw, L. Wirz, and L. Hak, "An efficient medical records access control with auditable outsourced encryption and decryption," in *Proc. 15th Int. Conf. Knowl. Smart Technol. (KST)*, Feb. 2023, pp. 1–6, doi: 10.1109/KST57286.2023.10086904.

[25] S. Alshehri, O. Bamasaq, D. Alghazzawi, and A. Jamjoom, "Dynamic secure access control and data sharing through trusted delegation and revocation in a blockchain-enabled cloud-IoT environment," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 4239–4256, Mar. 2023, doi: 10.1109/JIOT.2022.3217087.

[26] O. Cheikhrouhou, K. Mershad, F. Jamil, R. Mahmud, A. Koubaa, and S. R. Moosavi, "A lightweight blockchain and fog-enabled secure remote patient monitoring system," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100691.

[27] S. Rahmadika, P. V. Astillo, G. Choudhary, D. G. Duguma, V. Sharma, and I. You, "Blockchain-based privacy preservation scheme for misbehavior detection in lightweight IoMT devices," *IEEE J. Biomed. Health Informat.*, vol. 27, no. 2, pp. 710–721, Feb. 2023, doi: 10.1109/JBHI.2022.3187037.

[28] F. Daidone, B. Carminati, and E. Ferrari, "Blockchain-based privacy enforcement in the IoT domain," *IEEE Trans. Depend. Secure Comput.*, vol. 19, no. 6, pp. 3887–3898, Nov. 2022, doi: 10.1109/TDSC.2021.3110181.

[29] PYCA. Python Cryptographic Authority. (Nov. 6, 2022). *Pyca/Cryptography GitHub*. Accessed: Nov. 7, 2022. [Online]. Available: github.com/pyca/cryptography

[30] J. Bethencourt. (May 2006). *Advanced Crypto Software Collection*. Acsc.cs.utexas.edu. Accessed: Nov. 7, 2022. [Online]. Available: acsc.cs.utexas.edu/cpabe/

[31] *PBC (Pairing-Based Cryptography) Library*. Accessed: Oct. 14, 2022. [Online]. Available: https://crypto.stanford.edu/pbc/

[32] Ethereum, Documentation for Remix IDE. (May 26, 2023). *Ethereum/Remix-Ide*. GitHub. Accessed: May 26, 2023. [Online]. Available: github.com/ethereum/remix-ide

[33] Conancom. (May 26, 2023). *Secure and Lightweight Blockchain-enabled Access Control for Fog-Assisted IoT Cloud based Medical Treatment Records Sharing*. Conancom/Light-Med. GitHub. Accessed: May 27, 2023. [Online]. Available: github.com/conancom/light-med

SOMCHART FUGKEAW (Member, IEEE) received the bachelor's degree in management information systems from Thammasat University, Bangkok, Thailand, the master's degree in computer science from Mahidol University, Thailand, and the Ph.D. degree in electrical engineering and information systems from The University of Tokyo, Japan, in 2017. He is currently an Assistant Professor with the Sirindhorn International Institute of Technology, Thammasat University. His research interests include information security, access control, cloud computing security, big data analysis, and high-performance computing. He has served as a Reviewer for several international journals, such as IEEE Access, the IEEE Transactions on Information Forensics and Security, the IEEE Transactions on Dependable and Secure Computing, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Big Data, the IEEE Transactions on Network Science and Engineering, the IEEE Transactions on Network and Service Management, the IEEE Transactions on Parallel and Distributed Systems, *Computer & Security*, the IEEE Systems Journal, and *ACM Transactions on Multimedia Computing, Communications, and Applications*.

LEON WIRZ is currently pursuing the bachelor's degree in computer engineering with the Sirindhorn International Institute of Technology, Thammasat University. His research interests include cloud computing, network security, and information security.

LYHOUR HAK is currently pursuing the bachelor's degree in computer engineering with the Sirindhorn International Institute of Technology, Thammasat University. His research interests include network security, blockchain, and information security.