IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 16, 2017, accepted January 10, 2018, date of publication January 15, 2018, date of current version February 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2793304

Privacy-Preserving Search Over Encrypted Personal Health Record In Multi-Source Cloud

XIN YAO[1,2], (Student Member, IEEE), YAPING LIN[1,2], (Member, IEEE), QIN LIU[1], AND JUNWEI ZHANG[3], (Member, IEEE)

[1]College of Computer Science and Electronic Engineering, Hunan University, Changsha 410006, China
[2]Hunan Provincial Key Laboratory of Dependable Systems and Networks, Changsha 410082, China
[3]School of Cyber Engineering, Xidian University, Xi'an 710126, China

Corresponding author: Yaping Lin (yplin@hnu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61472125, Grant 61402161, Grant 61632009, and Grant 61472310, and in part by the China Scholarship Council.

ABSTRACT Cloud-based Personal Health Record systems (CB-PHR) have great potential in facilitating the management of individual health records. Security and privacy concerns are among the main obstacles for the wide adoption of CB-PHR systems. In this paper, we consider a multi-source CB-PHR system in which multiple data providers, such as hospitals and physicians are authorized by individual data owners to upload their personal health data to an untrusted public cloud. The health data are submitted in an encrypted form to ensure data security, and each data provider also submits encrypted data indexes to enable queries over the encrypted data. We propose a novel Multi-Source Order-Preserving Symmetric Encryption (MOPSE) scheme whereby the cloud can merge the encrypted data indexes from multiple data providers without knowing the index content. MOPSE enables efficient and privacy-preserving query processing in that a data user can submit a single data query, the cloud can process over the encrypted data from all related data providers without knowing the query content. We also propose an enhanced scheme, MOPSE$^+$, to more efficiently support the data queries by hierarchical data providers. Extensive analysis and experiments over real data sets demonstrate the efficacy and efficiency of MOPSE and MOPSE$^+$.

INDEX TERMS Authorization query, cloud computing, personal health record, privacy-preserving query.

I. INTRODUCTION

Cloud-based Personal Health Record systems (CB-PHR) such as Microsoft HealthVault[1] and ZebraHealth[2] are rising. A typical CB-PHR system consists of three entities: data owners, data providers and a cloud server. In CB-PHR system, data owners and data providers are defined as patients themselves and hospitals, respectively. Data owners can directly authorize data providers to upload their PHRs to the cloud. The CB-PHR system allows data owners to access their PHRs anytime and anywhere, be better prepared for medical appointments and unexpected emergencies, maintain a more complete picture about personal health, and even achieve fitness goals. Data providers can explore the CB-PHR system to provide better medical services by sharing, collaborating, and engaging with the patients in new ways.

Privacy concerns are among the main obstacles for the wide adoption of CB-PHR systems. In particular, many people have deep concerns that there can be unauthorized access to their sensitive PHRs. For example, the cloud may have business interest in analyzing the PHRs, and it may also have malicious employees or even be hacked. A natural way to alleviate the privacy concerns is to let data owners and providers upload encrypted PHRs to the cloud which does not possess the decryption keys [1]–[5], [8]. Since PHRs can be in huge volume, it is very inefficient for data owners or providers to retrieve all the encrypted PHRs from the cloud when only a small portion of them are needed. To enable efficient queries over encrypted PHRs, the B^+-tree technique [2]–[5], [9] is proposed to build an index for each patient's PHRs. The data index allows the cloud server to quickly find all the PHRs matching a particular data query. To further resolve the privacy concerns about data indexes and queries, searchable encryption schemes [10]–[20] are proposed to encrypt data indexes and queries as well. These schemes allow the cloud server to perform efficient queries over encrypted PHRs directly based on the encrypted indexes and queries while blind to the index and query content.

[1]http://www.healthvault.com
[2]https://www.zebrahealth.com

Traditional searchable encryption schemes [10]–[20] are designed for generic cloud platforms and not optimized for CB-PHR systems. In particular, the PHRs of different data providers for the same data owner may be highly correlated and associated with the same attributes (e.g., symptoms). If a traditional search encryption scheme is used, each data provider needs to independently generate the encrypted data index for submission to the cloud server. Therefore, the data owner needs to manage the keys with different data providers and also submit a dedicated data query for each data provider even if query conditions are exactly the same. A plausible solution to this issue is to let all the data providers use a common key assigned by the data owner to encrypt the data indexes associated with him.[3] This method, however, is vulnerable to the compromise of a single data provider.

In this paper, we propose a very efficient PHR system with strong privacy guarantees. In our system, each data owner authorizes multiple data providers to submit encrypted health records and data indexes to the cloud server. Our system differs from prior work in two desirable features. First, each data provider of the same data owner uses a unique symmetric key for encrypting data indexes, thus resisting single point of compromise. Second, each data owner needs not manage the keys with individual health providers and can submit a single encrypted query to the cloud server for searching over the encrypted health data from all his data providers. The second feature enables very efficient query processing.

Our system is built upon Multi-source Encrypted Indexes Merge (MEIM), a novel technique we propose in this paper. MEIM allows the cloud server to merge multiple encrypted data indexes from different health providers of the same patient without violating the patient's privacy. It also permits the patient to generate a single encrypted query over all his health providers' encrypted data stored at the cloud server. The fundamental building block of MEIM is a novel Multi-source Order-Preserving Symmetric Encryption (MOPSE) primitive we develop. MOPSE preserves the order of multiple data indexes encrypted by different symmetric keys.

We also propose an MOPSE$^+$ primitive to support hierarchical authorization queries whereby the health providers with higher privileges can query the cloud server for the encrypted data from those with lower privileges. Such hierarchical access patterns are quite common in practice.

We confirm the security and efficiency of our system by comprehensive theoretical analysis and extensive experiments with a real dataset [21]. Our results show that (1). the query performance for data users in MOPSE and MOPSE$^+$ is faster $n\times$[4] than that in tradition OPSE; (2). the query performance for data providers in MOPSE and OPSE are almost the same and less than that in MOPSE$^+$.

The rest of this paper is organized as follows: Section II presents models and problem statement. Section III gives the details for MOPSE scheme. The construction of MOPSE$^+$ scheme is shown in Section IV. Section V and VI present the performance and security analysis. Section VII presents the experiment evaluation. Finally, we conclude this paper in Section IX, after presenting related work in Section VIII.

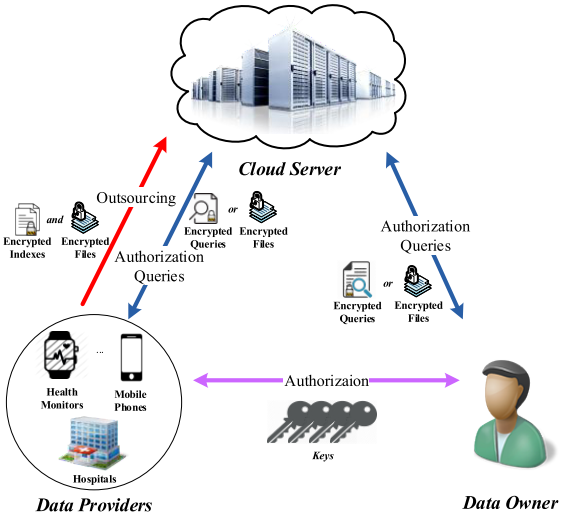

FIGURE 1. System model.

II. SYSTEM AND THREAT MODELS

A. SYSTEM MODEL AND WORKFLOW

We consider a generic CB-PHR system shown in Fig. 1. There are three kinds of entities: the cloud server, data owners, and data providers. A data owner refers to a patient who own the PHRs. In contrast, a data provider can refer to a patient himself, any of his health providers such as a physician or hospital, and even his personal health monitoring device. The cloud server stores and provides anytime, anywhere access to the PHRs submitted by the data providers of each data owner.

Each data owner has strong privacy concerns for his PHRs. His data providers thus must encrypt the PHRs before outsourcing them to the cloud server. To ensure efficient search for the encrypted PHRs, each data provider additionally uploads a data index to the cloud server. The data owner or any of his authorized data providers can submit data queries to the cloud server. Both data indexes and queries should be encrypted as well to prevent information disclosure. The cloud server explores the data indexes to locate the PHRs satisfying each query without the capability or need to decrypt the PHRs, data indexes, or data queries. Finally, the cloud server returns the corresponding encrypted PHRs to the requesting data user who can decrypt them with the right decryption key.

B. THREAT MODEL

We assume a conventional threat model as follows. The cloud server is honest-but-curious by faithfully running the system but having strong interest in the content of the PHRs, data indexes, and queries. We also assume that data providers

[3]No gender implication.
[4]Here, n denotes the number of data providers

TABLE 1. Notation.

Symbol	Definition
R and r_d	Integer range set and integer range (l_d,u_d)
n	MDBTs numbers
ψ	Attribute numbers in original index
γ	Health records numbers
φ	A large prime
$\mathbb{G}_j$, j={1,2,$\mathcal{T}$}	Three cyclic groups of order φ
g_j, j={1,2,$\mathcal{T}$}	The group element of $\mathbb{G}_j$, j={1,2,$\mathcal{T}$}
$\mathbb{V}$ and $\mathbb{V}^*$	Two vector spaces
$\mathbf{x}$ and $\mathbf{y}$	Two vectors $(g_1^{x_1},\cdots,g_1^{x_n})$ and $(g_2^{y_1},\cdots,g_2^{y_n})$
$\mathbb{A}$ and $\mathbb{A}^*$	Two canonical bases of $\mathbb{V}$ and $\mathbb{V}^*$
$\vec{x}/\vec{v}$	Plaintext/Predicate vector

are untrusted and may try to acquire the PHRs generated by other providers. Besides, we assume that the communications within our system are secured using traditional mechanisms such as TLS (Transport Layer Security) [22]. Finally, we assume that the cloud server does not collude with data providers to compromise data owners' privacy.

The privacy of data owners can be classified into PHR privacy, index privacy, and query privacy. Since our system stores encrypted PHRs at the cloud server which has no decryption keys, PHR privacy is easily achieved as long as the underlying encryption primitive is unbreakable. We thus focus on index privacy and query privacy hereafter. Index privacy is considered compromised if the content of any encrypted index is known to the cloud server or any data provider other than the source data provider. In contrast, query privacy is said to be breached in either scenario below. First, the content of any encrypted query is disclosed to anyone other than the data user (i.e., the data owner or any of his authorized data providers). Second, a data provider generates a valid query without obtaining the authorization of the data owner.

III. MULTI-SOURCE ENCRYPTED INDEX MERGING

Our system features a novel Multi-source Encrypted Index Merging (MEIM) technique whereby the cloud server can merge the encrypted data indexes from different data providers of the same data owner without decrypting individual indexes. MEIM allows a data user to submit a single data query for the PHRs from all the data providers of the data owner. In what follows, we introduce the basics of indexing and querying PHRs, followed by the MEIM design.

A. INDEXING AND QUERYING PHRs

We explore the classic multi-dimensional *B*-tree (MDBT) [9] to index PHRs. In particular, we assume that each PHR has the same set of queryable attributes such as name, year, disease, and blood pressure. The values of each attribute can be naturally numerical (e.g., year) or non-numerical (e.g., disease), and we convert non-numerical attribute values into numerical ones with the coding technique in [23]. Each attribute corresponds to one layer on the MDBT, and each node (the root, non-leaf nodes, and leaf nodes) has one or multiple attribute values depending on the particular PHRs. In addition, each leaf node has a pointer to the corresponding encrypted PHR. Fig. 2 shows an exemplary MDBT index for PHRs with three attributes: year, disease, and blood pressure.

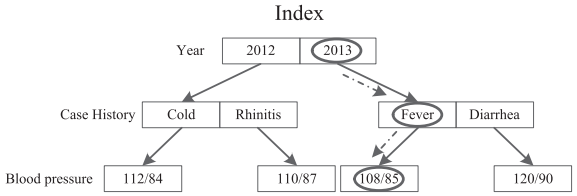

FIGURE 2. An MDBT index example with three attributes, year, case history and blood pressure. The nodes satisfying the query are marked with black circles, and the arrows show the matching processing.

The MDBT index supports multi-attribute equality, subset, and range queries. For example, a query related to Fig. 2 can be $(2012 \leq \mathsf{year} \leq 2013) \wedge (\mathsf{disease} \in \{Cold, Fever\}) \wedge (\mathsf{bloodpressure} = 108/85)$, where $\wedge$ denotes the conjunction operator. The sub-queries for the year, disease, and bloodpressure attributes correspond to range, subset, and equality queries, respectively. The nodes matching this query in each layer are marked with black circles in Fig. 2. The leaf node with value 108/85 points to the PHR that should be returned to the data user. Equality queries are specific range queries, and subset queries can be easily transformed into range or equality queries. So we focus on range queries hereafter.

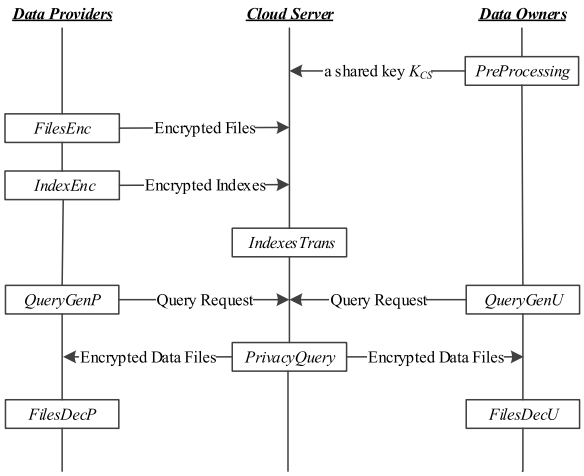

FIGURE 3. This figure shows the MEIM workflow. The details are represented in Section III-B.

B. OVERVIEW OF MEIM

Fig. 3 depicts the work flow of MEIM with the following function modules for each involved entity.

Data Owner: (1) *PreProcessing* generates a group key K_{DP}, a secret key K kept to itself, and a secret key K_{CS} shared with the cloud server; (2) *QueryGenU* generates his

data queries; and (3) *FilesDecU* decrypts the encrypted PHRs returned by the cloud server.

Data Provider: (1) *FilesEnc* encrypts the PHRs with a symmetric-key algorithm and then encrypts the symmetric key with a public-key algorithm; (2) *IndexEnc* creates the MDBT index and then encrypts it with our Multi-source Order-Preserving symmetric encryption (MOPSE) scheme; (3) *QueryGenP* generates its data queries; and (4) *FilesDec* decrypts the encrypted PHRs returned by the cloud server.

Cloud Server: (1) *IndexTrans* merges multiple encrypted MDBT indexes and then segments the merged index into two indexes: a S-index and an H-index; (2) *PrivacyQuery* explores the S-index and H-index to process the encrypted queries and then returns the corresponding PHRs to data users or data providers.

C. MULTI-SOURCE ORDER-PRESERVING SYMMETRIC ENCRYPTION

We propose a novel MOPSE scheme for each data provider to encrypt the MDBT index to enable later merging at the cloud server. MOPSE is adapted from Order-Preserving Symmetric Encryption (OPSE) [24]. To introduce OPSE and MOPSE, we first introduce the following definition.

Definition 1: For $D_1, D_2 \subset \mathbb{N}$ with $|D_1| \leq |D_2|$, a function f: $D_1 \rightarrow D_2$ is order-preserving (aka. strictly increasing) if $\forall i, j \in D_1, f(i) > f(j)$ iff $i > j$.

Consider a deterministic encryption scheme $\{\mathcal{K}, \mathsf{Enc}(K, \bullet), \mathsf{Dec}(K, \bullet)\}$ with the key space $\mathcal{K}$, the plaintext space $\mathcal{D}$, the ciphertext space $\mathcal{R}$, the encryption function $\mathsf{Enc}(K, \bullet)$, and the decryption function $\mathsf{Dec}(K, \bullet)$. We call this scheme an OPSE scheme iff $\mathsf{Enc}(K, \bullet)$ is an order-preserving function from $\mathcal{D}$ to $\mathcal{R}$ for $\forall K \in \mathcal{K}$. OPSE assumes a single data source with the same key K, so it is not applicable to our multi-source scenario in which different data providers use different keys to encrypt the MDBT indexes for the same data owner. More specifically, assume that two data providers use keys K_1 and K_2, respectively. We then have $\mathsf{Enc}(K_1, d_1) < \mathsf{Enc}(K_1, d_2)$ and $\mathsf{Enc}(K_2, d_1) < \mathsf{Enc}(K_2, d_2)$ for $\forall d_1 < d_2$. But $\mathsf{Enc}(K_2, d_2)$-$\mathsf{Enc}(K_1, d_1)$ can be zero, positive or negative. We thus propose MOPSE to preserve the order of ciphertexts encrypted with various symmetric keys. For the above example, $\mathsf{Enc}(K_2, d_2)$ is larger than $\mathsf{Enc}(K_1, d_1)$ in MOPSE. More formally, we define the property of MOPSE as follows.

Definition 2: Given a set of OPSE schemes $\{\mathsf{Enc}(K_1, \bullet), \cdots, \mathsf{Enc}(K_n, \bullet)\}$ with the common plaintext space $\mathcal{D}$ and ciphertext space $\mathcal{R}$. This set is called an MOPSE set iff

$$\mathsf{Enc}(K, d_i) < \mathsf{Enc}(K', d_j), \quad \forall K, K' \in \{K_i\}_{i=1}^{n}, \forall d_i < d_j \in \mathcal{D}. \tag{1}$$

We further introduce attribute-specific data ranges for the MOPSE design. Consider a data owner with n data providers and an arbitrary attribute with an integer value in [min, max]. In the *PreProcessing* phase, the data owner generates a data-range set R_i of size $\max - \min + 1$ for his ith data provider, where $R_i = \{[l_{i,d}, u_{i,d}]\}_{d=\min}^{\max}$ with $[l_{i,d}, u_{i,d}]$ as a continuous set of non-negative integers. The n data-range sets satisfy three conditions. First, the data range of any provider i for any attribute value is always to the left of that for any higher attribute value: $u_{i,d_1} < l_{i,d_2}$, $\forall d_1 < d_2 \in [\min, \max], \forall i \in [1, n]$. Second, the data providers have no overlapping data ranges for any attribute value: $[l_{i,d}, u_{i,d}] \cap [l_{j,d}, u_{j,d}] = \phi, \forall i \neq j \in [1, n], \forall d \in [\min, \max]$. Last, the data ranges of all the n providers for the same attribute value are to the left of those for any higher attribute value: $u_{i,d_1} < l_{j,d_2}, \forall d_1 < d_2 \in [\min, \max]$, $\forall i, j \in [1, n]$. Fig. 4 shows an example with two data providers and two attributes (1 and 2). Based on the data ranges, the data owner further generates a mapping function f_i for each data provider i as follows.

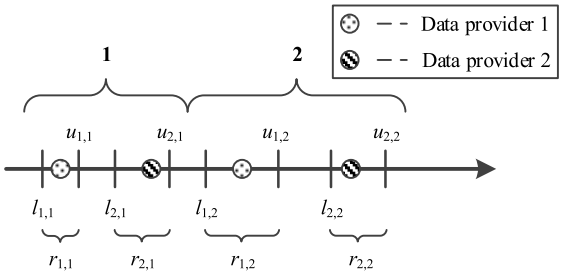

FIGURE 4. An example of data range $R=\{R_1, \cdots, R_n\}$. All ciphertexts for 1 locate at the left of that for 2.

Definition 3: Given any integer $d \in [\min, \max]$, the corresponding ciphertext C_d, and a data-range set $R_i = \{[l_{i,d}, u_{i,d}]\}_{d=\min}^{\max}$, the mapping function $f_i(\cdot)$ for data provider $i \in [1, n]$ is defined as

$$f_i(d) = l_{i,d} + (C_d \bmod (u_{i,d} - l_{i,d})). \tag{2}$$

Our construction also explores the Prefix Membership Verification (PMV) technique in [25] which is defined as follows.

Definition 4: For an integer d with a w-bit binary representation, $b_1 b_2 \ldots b_w$ (b_i=0,1), the function $\mathcal{F}(d)$=$\{b_1 b_2 \ldots b_w, b_1 b_2 \ldots b_{w-1}*, \ldots, b_1 * \cdots *, ** \cdots *\}$ is defined as the prefix family of d, where $*$ is the wildcard character, and $|\mathcal{F}(d)| = w + 1$.

For example, the prefix family of 9 is $\mathcal{F}(9)$=$\mathcal{F}(1001) = \{1001, 100*, 10**, 1***, ****\}$.

Definition 5: $\mathcal{S}([d_1, d_2])$ denotes the minimum set of prefixes for the integer range $[d_1, d_2]$ for any $d_1 \leq d_2$.

For instance, $\mathcal{S}([12, 15]) = \{11**\}$, $\mathcal{S}(11) = \{1011\}$, and $\mathcal{S}([11, 15]) = \{1011, 11**\}$.

Definition 6: $\mathcal{N}(\cdot)$ is called a prefix numericalization function if it satisfies two properties: (1) for any prefix d, $\mathcal{N}(d)$ is a binary string; (2) for any two prefixes d_1 and d_2, $\mathcal{N}(d_1)$=$\mathcal{N}(d_2)$ iff d_1=d_2.

We adopt the prefix numericalization scheme in [26] defined as follows. For a w-bit prefix $d = b_1 b_2 \cdots b_k * \cdots *$ with $w - k$ wildcard characters, we compute $\mathcal{N}(d)$ by inserting 1 after b_k and then replacing all $*$'s to 0s. For example, if $d = 10**$, $\mathcal{N}(d)$=10**100**; if $d = 1001$, $\mathcal{N}(d)$=1001**1**.

Another building block we use is an order-preserving minimal perfect hashing (OPMPH) function defined as follows [27].

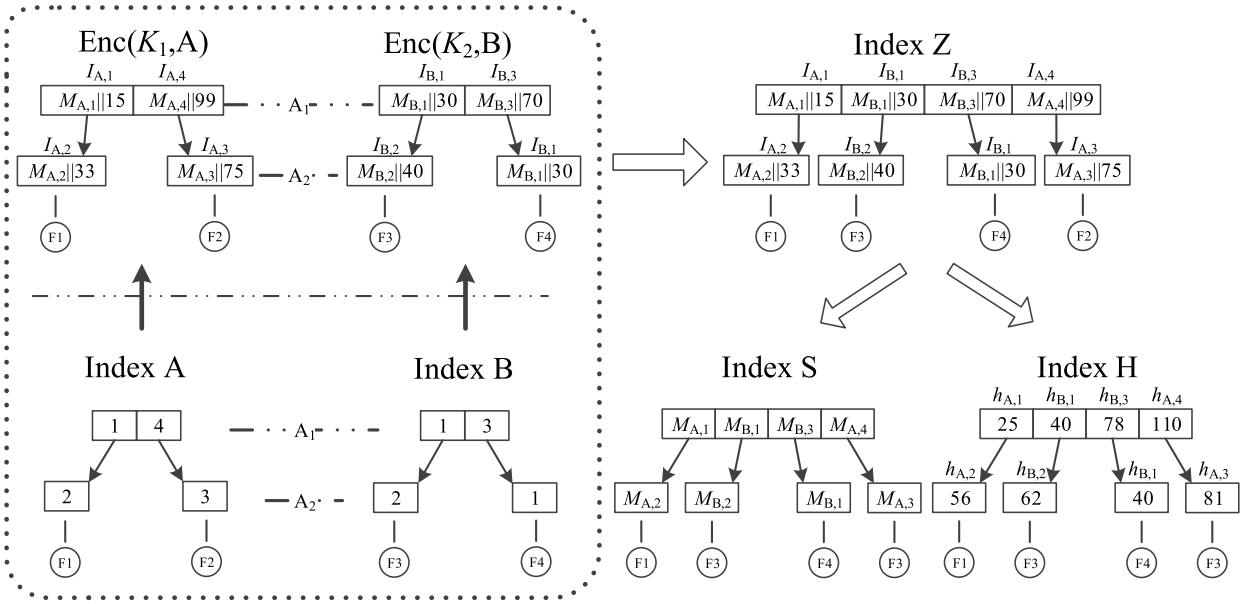

FIGURE 5. An example of MEIM. The dashed part is processed by two different data providers, who generate indexes A and B, and encrypt indexes to Enc(K_1,A) and Enc(K_2,B). The cloud server runs the other part, e.g., merging indexes Enc(K_1,A) and Enc(K_2,B) to Z and segmenting Z into two indexes S and H. Here $I_{i,d} = \{M_{i,d}||P_{i,d}\}$ denotes the ciphertext of the value d for index i.

Definition 7: Given a data set D with a total (numerical) order, an OPMPH function for $\forall d_1, d_2 \in D$ *such that if* $d_1 = d_2 \rightarrow \mathcal{H}_c(d_1) = \mathcal{H}_c(d_2)$ *and* $d_1 < d_2 \rightarrow \mathcal{H}_c(d_1) < \mathcal{H}_c(d_2)$.

Now we are ready to present MOPSE which consists of the following four sub-algorithms.

(1) **Setup**. Given the group key $\mathcal{K}_{\mathsf{DP}}$ and the security key K, the data owner outputs an OPMPH function $\mathcal{H}_c(\cdot)$, a mapping function $f_i(\cdot)$ as in Eq. (2), a secret key K_i, and a data-range set R_i for each data provider $i \in [1, n]$. Note that $\mathcal{H}_c(\cdot)$ is the same for all the data providers.

(2) **CiphertextGen**. Data provider i runs a symmetric encryption algorithm (e.g., AES [28]) with the key K_i to produce the ciphertext $C_{i,d}$ of an attribute value d.

(3) **LabelGen**. Upon generating $C_{i,d}$, data provider i deploys the mapping function f_i and the OPMPH function $\mathcal{H}_c$ to output a ciphertext $P_{i,d}$. Here, we define $P_{i,d}$ by the value $\mathcal{H}_c(f_i(C_{i,d})) \in \mathcal{H}_c(r_{i,d})$. Where $\mathcal{H}_c(r_{i,d})$ denotes an range $(\mathcal{H}_c(l_{i,d}), \mathcal{H}_c(u_{i,d}))$;

(4) **Labelblind**. Given a ciphertext $P_{i,d}$, it runs the prefix family function $\mathcal{F}$ to output the prefixes $\mathcal{F}(P_{i,d})$. The numericalization function $\mathcal{N}$ numericalizes each prefix independently. We abuse the notation by letting $\mathcal{N}(\mathcal{F}(P_{i,d}))$ denote the outputs. Subsequently, data provider i computes the keyed-Hash Message Authentication Code (HMAC) of each numericalized prefix using the secret key K_i, which is denoted by $M_{i,d} = \text{HMAC}_{K_i}(\mathcal{N}(\mathcal{F}(P_{i,d})))$. Note that HMAC can be either HMAC-MD5 or HMAC-SHA1, and it satisfies the following properties: the one-wayness and the collision resistance. Finally, it concatenates $M_{i,d}$ and $P_{i,d}$ as the final label $I_{i,d} = \{M_{i,d}||P_{i,d}\}$, where $||$ denotes the concatenation.

Correctness: The label $I_{i,d}$ of the value d for data provider i consists of $M_{i,d}$ and $P_{i,d}$. Given two arbitrary value d_1, $d_2 \in \mathbb{N}$ with $d_1 < d_2$, two MOPSE schemes MOPSE(K_1, •) and MOPSE(K_2, •) are utilized to encrypt them. The values P_{1,d_1} and P_{2,d_1} for d_1 belong to $(\mathcal{H}_c(l_{1,d_1}), \mathcal{H}_c(u_{1,d_1}))$ and $(\mathcal{H}_c(l_{2,d_1}), \mathcal{H}_c(u_{2,d_1}))$, while the values P_{1,d_2} and P_{2,d_2} for d_2 locate in the ranges $(\mathcal{H}_c(l_{1,d_2}), \mathcal{H}_c(u_{1,d_2}))$ and $(\mathcal{H}_c(l_{2,d_2}), \mathcal{H}_c(u_{2,d_2}))$, respectively. Since d_1 is less than d_2, the maximum value $\text{Max}\{P_{1,d_1}, P_{2,d_1}\}$ is less than the minimum value $\text{Min}\{P_{1,d_2}, P_{2,d_2}\}$. Hence, the construction of the MOPSE satisfies Def. 2.

D. GENERATING ENCRYPTED INDEX

In the following subsections, we depict how the MEIM mechanism runs by an example in Fig. 5. First, data providers generate platintext indexes with a specific strucutre MDBT, e.g., indexes A and B in Fig. 5. Subsequently, data providers utilize MOPSE to generate encrypted indexes Enc(K_1,A) and Enc(K_2,B). Here, each label $I_{i,d}$ is represented by $\{M_{i,d}||P_{i,d}\}$. For instance, the label $I_{B,1}$ of the value 1 for data provider B is $\{M_{B,1}||30\}$, where $M_{B,1}$ is a prefix family (each prefix is a 128-bit hash key) and $P_{B,1}$ is 30. Finally, all these encrypted indexes are outsourced to the cloud server.

E. TRANSFORMING ENCRYPTED INDEXES

Upon receiving encrypted indexes, the cloud server recursive merges encrypted indexes from the root to leaf nodes. Here, MEIM compares $P_{i,d}$ values to determine the orders of nodes in the merged index. In our example, the index Z in Fig. 5 is a merged index from Enc(K_1,A) and Enc(K_2,B). $P_{A,1}$ (15) in Enc(K_1,A) index is less than $P_{B,3}$ (70) in Enc(K_2,B), so $I_{A,1}$ priors to $I_{B,3}$ in Z. Subsequently, the cloud server segments

the merged index to two indexes: S index and H index. As shown in Fig. 5, the $M_{i,d}$ in S index is equivalent to that in Z, while $h_{i,d}$ in H index is generated by encoding $P_{i,d}$ in Z with the OPMPH function $\mathcal{H}_s$, which satisfies Def. 7 and is generated by the shared key K_{CS}. As illustrated by the previous example, the labels $I_{A,3}$ and $I_{B,3}$ in Z are $\{M_{A,3}||75\}$ and $\{M_{B,3}||70\}$, respectively. Hence, the order of $M_{A,3}$ and $M_{B,3}$ in S index follows that in Z. Meanwhile, if let $\mathcal{H}_s(70)$ be 78, $h_{B,3}$ in H is 78.

F. TRAPDOOR GENERATION

In the trapdoor generation, it is divided into two categories: for data owners (QueryGenU) and for data providers (QueryGenP). For brevity, we first consider single attribute trapdoor generation, and then describe how to extend it to multi-attribute trapdoor generation.

Given a query condition q, data owners first determine the upper bound $\sup_q$ and the lower bound $\inf_q$ in R. Namely, $\sup_q$ and $\inf_q$ are $\text{Max}\{u_{*,q}\}$ and $\text{Min}\{l_{*,q}\}$, respectively. To generate a trapdoor securely, data owners should generate different trapdoor each time. Thus, we introduce two disturbing parameters ι_1 and ι_2 to a query pair $\hat{q} = \{\inf_q + \iota_1, \sup_q + \iota_2\}$. Finally, data owners calculate the final hash value $\tilde{q} = \mathcal{H}_s(\mathcal{H}_c(\hat{q}))$ with the functions $\mathcal{H}_c$ and $\mathcal{H}_s$, successively. Without loss of generality, we also consider range query, and define q as a range. In this case, we first find the $q_{\min}$ and $q_{\max}$ in the range q, and then replace $\text{Max}\{u_{*,q}\}$ and $\text{Min}\{l_{*,q}\}$ with $\text{Max}\{u_{*,q_{\max}}\}$ and $\text{Min}\{l_{*,q_{\min}}\}$. While for the trapdoor generation of the subset, we can transform it to its corresponding equality or range, as described in Section III-A. Note that, to generate multi-attribute trapdoor $Q = \{q_1, \cdots, q_\psi\}$, we generate the final trapdoor $\tilde{q}_j$ for each q_j ($j \in [1, \psi]$), and output the final trapdoor $\widetilde{Q} = \{\tilde{q}_1, \cdots, \tilde{q}_\psi\}$.

Next, we discuss how to set the disturbing parameters ι_1 and ι_2. Since $\inf_q > \sup_{q-1}$ and $\sup_q < \inf_{q+1}$, the disturbing parameters ι_1 and ι_2 can be random chosen in the range $[0, |\inf_q - \sup_{q-1}|]$ and $[0, |\inf_{q+1} - \sup_q|]$, respectively.

For a query condition q, data provider i first encrypts it with the secret key K_i to obtain $\widehat{C}_{i,q}$, and computes the hash value $\widehat{P}_{i,q} = \mathcal{H}_c(\widehat{C}_{i,q})$. Then, data provider outputs the trapdoor $\tilde{t} = \text{HMAC}_{K_i}(\mathcal{N}(\mathcal{S}(\widehat{P}_{i,q})))$. If q is a range, we only need to transform $\widehat{P}_{i,q}$ to a range. Likewise, we can also utilize $\widetilde{T} = \{\tilde{t}_1, \cdots, \tilde{t}_\psi\}$ to address the case of the multi-attribute.

G. PRIVACY-PRESERVING QUERY

After receiving the trapdoor $\widetilde{Q}$ issued by data owners, the cloud server iterative processes the query on H index from the root to the leaf nodes as shown in Alg. 1. If the query fails, the algorithm returns **Null**; Otherwise, the encrypted files. Likewise, for the trapdoor $\widetilde{T}$ from data provider, the matching is also an iterative as depicted in Alg. 1. Only the matching condition in Line 4 of Alg. 1 follows the Theorem 1, which is proof in [29].

Algorithm 1 *PrivacyQuery*

Input: (H, $\widetilde{Q}$) or (S, $\widetilde{T}$)
Output: Encrypted files
1 Extract 1^{st} query attribute value in $\widetilde{Q}$ or $\widetilde{T}$ as $\tilde{q}$ or $\tilde{t}$;
2 Remove 1^{st} query attribute value in $\widetilde{Q}$ or $\widetilde{T}$;
3 **for** *Traverse each element* χ *in H or S index from left to right* **do**
4 **if** $\chi.P \in \tilde{q}$ *or* $\chi.M \cap \tilde{t} \neq \phi$ **then**
5 **if** χ *belongs to a Leaf node* **then**
6 **return** *Encrypted file*;
7 **else**
8 *PrivacyQuery*(H.childnode, $\widetilde{Q}$) or *PrivacyQuery*(S.childnode, $\widetilde{T}$);

Theorem 1: Given a query range [a,b] and the attribute value d, d∈[a,b] if and only if the following formula is true.

$$HMAC_{K_i}(\mathcal{N}(\mathcal{F}(d))) \cap HMAC_{K_i}(\mathcal{N}(\mathcal{S}([a,b]))) \neq \phi \quad (3)$$

IV. ENHANCED SCHEME

In previous model, data providers (e.g., the hospitals and personal health monitors) can only access the patient's data issued by themselves. However, in reality, data providers may access health data generated by other data providers. For instance, research-oriented hospitals utilize patients' data to prevent the incidence of common diseases; the doctors may access the data recorded by personal health devices. Thus, considering a hierarchical system model will make our model more practical. In our hierarchical model, data providers have various privileges according to the actual requirements. For example, personal health devices are mostly with the lowest privileges, because of being accessed in common; research-oriented hospitals are with higher privileges than community hospitals, since the formers need amount of patients' data to do research, while community hospitals only access specific patients' data.

With the basic solution MOPSE, we can implement hierarchical structure for data providers, i.e., issuing all keys of data providers with lower privileges to those with higher privileges. However, the solution is inefficient, such as the overhead of the management of keys increases linearly with the number of data providers with lower privileges, and data providers need to generate amount of queries (the number is positively related to providers' number). To thwart these inefficient problems, we proposed an enhanced scheme (named MOPSE$^+$) based on our previous scheme HPBPE [30].

A. PRELIMINARIES

In this subsection, we depict some cryptographic definitions and assumptions, which follow our previous paper [30].

Assume that $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_\mathcal{T}$ are three cyclic groups with identical prime order φ. Let g_j be the generator of $\mathbb{G}_j$ ($j \in [1, 2]$). We define a non-degenerate bilinear pairing operation e as $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathcal{T}$, and $e(g_1, g_2) = g_\mathcal{T} \neq 1$. Notice that the multiplication operations are in three cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_\mathcal{T}$, which bear symmetric pairing groups ($\mathbb{G}_1 = \mathbb{G}_2$). In what follows, we describe some definitions as follows:

Definition 8 (Vector Spaces $\mathbb{V}$ and $\mathbb{V}^*$): Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two N-dimensional vectors. $\mathbb{V} = \underbrace{\mathbb{G}_1 \times \cdots \times \mathbb{G}_1}_{N}$ *and* $\mathbb{V}^* = \underbrace{\mathbb{G}_2 \times \cdots \times \mathbb{G}_2}_{N}$ *be two vector spaces. Two vectors x and y in $\mathbb{V}$ and $\mathbb{V}^*$ are represented by as $(g_1^{x_1}, \cdots, g_1^{x_N})$ and $(g_2^{y_1}, \cdots, g_2^{y_N})$, respectively. Notice that $x_j, y_j \in \mathbb{F}_\varphi$ for $j \in [1, N]$.*

Definition 9 (Canonical Bases $\mathbb{A}$ and $\mathbb{A}^*$): Let $\mathbb{A} = (a_1, \cdots, a_N)$ and $\mathbb{A}^* = (a_1^*, \cdots, a_N^*)$ be the canonical bases of $\mathbb{V}$ and $\mathbb{V}^*$, respectively. Therein, $a_1 = (g_1, 1, \cdots, 1)$, $a_2 = (1, g_1, \cdots, 1), \cdots, a_N = (1, \cdots, 1, g_1)$, while $a_1^* = (g_2, 1, \cdots, 1)$, $a_2^* = (1, g_2, \cdots, 1), \cdots, a_N^* = (1, \cdots, 1, g_2)$.

Definition 10 (Pairing Operation): For $x \in \mathbb{V}$ and $y \in \mathbb{V}^*$, let the pairing operation $e(x, y) = \prod_{j=1}^{N} e(g_1^{x_j}, g_2^{y_j})$ be $e(g_1, g_2)^{\sum_{j=1}^{N} x_j \cdot y_j} = g_\mathcal{T}^{\vec{x} \cdot \vec{y}} \in \mathbb{G}_\mathcal{T}$.

Definition 11 (Dual Pairing Vector Spaces (DPVS)): Let the prime orders of elements in $(\varphi, \mathbb{V}, \mathbb{V}^*, \mathbb{G}_\mathcal{T}, \mathbb{A}, \mathbb{A}^*)$ be φ. The dimensional of vector spaces for $\mathbb{V}$ and $\mathbb{V}^*$ over $\mathbb{F}_\varphi$ are identical and equivalent to N. These elements and canonical bases $\mathbb{A}$ and $\mathbb{A}^*$ meet the following conditions:

1) *Non-degenerate bilinear pairing: The non-degenerate bilinear pairing $e: \mathbb{V} \times \mathbb{V}^* \to \mathbb{G}_\mathcal{T}$ is a polynomial-time computation. Namely, $e(a\boldsymbol{x}, b\boldsymbol{y}) = e(\boldsymbol{x}, \boldsymbol{y})^{ab}$ and if $e(\boldsymbol{x}, \boldsymbol{y}) = 1$ for all $x \in \mathbb{V}$, then $\boldsymbol{y} = \boldsymbol{0}$.*
2) *Dual orthonormal bases: For $a_j \in \mathbb{A}$ and $a_{j'}^* \in \mathbb{A}^*$, let the non-degenerate bilinear pairing $e(a_j, a_{j'}^*)$ be $g_\mathcal{T}^{\delta_{j,j'}}$ for $\forall j, j' \in [1, n]$. If $j = j'$, $\delta_{j,j'} = 1$, otherwise 0, and $g_\mathcal{T} \neq 1 \in \mathbb{G}_\mathcal{T}$.*
3) *Distortion maps: Let $\vartheta_{j,j'} \in \mathbb{V}$ and $\vartheta_{j,j'}^* \in \mathbb{V}^*$ be two polynomial-time computable endomorphisms. Therein, $\vartheta_{j,j'}(a_{j'}) = a_j$, $\vartheta_{j,j'}(a_k) = \boldsymbol{0}$ and $\vartheta_{j,j'}(a_j^*) = a_{j'}^*$, $\vartheta_{j,j'}(a_k^*) = \boldsymbol{0}$ if $k \neq j'$. $\vartheta_{j,j'}$ and $\vartheta_{j,j'}^*$ are denoted by "distortion maps".*

Definition 12 (Hierarchical Privilege-Based Predicate (HPBP)): For a positive integers tuple $\vec{\alpha} = (z, \ell; \alpha_1, \cdots, \alpha_\ell)$ $(\alpha_0 = 0 < \alpha_1 < \alpha_2 < \cdots < \alpha_\ell = z)$, let $\sum_t = \mathbb{F}_\varphi^{\alpha_\ell - \alpha_{\ell-1}} \setminus \{\vec{0}\} (j = 1, \cdots, \ell)$ be the set of privileges. We define $\sum = \cup_{j=1}^{\ell} (\sum_1 \times \cdots \times \sum_j)$ as the hierarchical privileges ($\sum_j \cap \sum_{j'} \neq \phi$, iff $j = j'$). For a hierarchical privilege $(\vec{x}_1, \cdots, \vec{x}_h) \in \sum$, let $f_{(\vec{v}_1, \cdots, \vec{v}_l)}$ be the hierarchical privilege-based predicate. Therein, $f_{(\vec{v}_1, \cdots, \vec{v}_h)}(\vec{x}_1, \cdots, \vec{x}_\ell) = 1$ iff $h \leq \ell$ and $\vec{x}_j \cdot \vec{v}_j = 0$ for $1 \leq j \leq \ell$.

B. MAIN DESIGN OF MOPSE$^+$

1) ENHANCED SYSTEM FRAMEWORK

MOPSE$^+$ consists of the following sub-algorithms.

(1) *Setup*$^+$. Given the group key $\mathcal{K}_{\mathsf{DP}}$ and the security key K, the data owner generates an OPMPH function $\mathcal{H}_c$, a symmetric key K_s, a mapping function f_i, and a data-range set R_i for each data provider $i \in [1, n]$. Notice that $\mathcal{H}_c$ and K_s are shared by all data providers. Subsequently, data owner generates the master key pair (pk, sk) with a security parameter 1^λ and a format of hierarchy $\vec{\mu}$ as inputs. Finally, given the master key pair and predicate vectors $\vec{V} = (\vec{v}_1, \cdots, \vec{v}_h)$, data owner issues a specific private key $sk_{(\vec{v}_1, \cdots, \vec{v}_h)}$.

(2) *Delegate*$^+$. To reduce the overhead of issuing keys of data owner, the MOPSE$^+$ scheme enables data providers with higher privileges to issue private key to their subordinates. The *Delegate*$^+$ algorithm takes the public key pk, the private key $sk_{(\vec{v}_1, \cdots, \vec{v}_h)}$ and the (h+1)th predicate vector $\vec{v}_{h+1}$ as inputs to issue the (h+1)th private key $sk_{(\vec{v}_1, \cdots, \vec{v}_h, \vec{v}_{h+1})}$.

(3) *CiphertextGen*$^+$ and *LabelGen*$^+$. Given an value d, *CiphertextGen*$^+$ generates two ciphertext $C_{i,d} = \mathsf{AES}(K_i, d)$ and $C_d = \mathsf{AES}(K_s, d)$. Then, *LabelGen*$^+$ generates $P_{i,d} = \mathcal{H}_c(f_i(C_{i,d}))$ with the mapping function f_i and the OPMPH function $\mathcal{H}_c$.

(4) *Labelblind*$^+$. Data provider i with hth privilege, takes the master public key pk, identification vectors $\vec{X}_h = (\vec{x}_1, \cdots, \vec{x}_h, \vec{x}_{h+1}^+, \cdots, \vec{x}_\ell^+)$ ($\vec{x}_j^+ \xleftarrow{U} \mathbb{F}_\varphi$ and h+1$\leq j \leq \ell$) and C_d as inputs, and returns the ciphertext $C_{h,d}$. Finally, $C_{h,d}$ is concatenated with $P_{i,d}$ as its label $I_{i,d} = \{C_{h,d} || P_{i,d}\}$, where $||$ denotes the concatenation.

2) SCHEME DETAILS

Here, we detail the construction of the MOPSE$^+$ scheme using dual pairing vector spaces (DPVS) [30]. Let $\mathbb{B} = (b_1, \cdots, b_{z+3})$ and $\mathbb{B}^* = (b_1^*, \cdots, b_{z+3}^*)$ be two vector spaces in DPVS, where $\mathbb{B}$ and $\mathbb{B}^*$ are both z+3 dimensional spaces. The public parameters for DPVS are $(b_1, \cdots, b_z, b_{z+1} + b_{z+2}, b_{z+3})$

(1) *Setup*$^+$. Taking 1^λ and z+3 as the input of $\mathcal{G}_{ob}$, it randomly selects the parameters param, $\mathbb{B}$ and $\mathbb{B}^*$ ((param, $\mathbb{B}$, $\mathbb{B}^*$) $\xleftarrow{R}$ $\mathcal{G}_{ob}(1^\lambda, z + 3)$). Notice that $(\vec{X}, \mathbb{B}^*)$ represents the master security key sk and (param, $\widehat{\mathbb{B}}$) denotes the master public key pk. Therein, $\widehat{\mathbb{B}}$ and $\vec{X}$ are expressed by $(b_1, \cdots, b_z, b_{z+1} + b_{z+2}, b_{z+3})$ and $(\vec{x}_1, \cdots, \vec{x}_\ell) = ((x_1, \cdots, x_{\mu_1}), \cdots, (x_{\mu_{\ell-1}+1}, \cdots, x_{\mu_\ell}))$, respectively. To generate the security key $sk_{(\vec{v}_1, \cdots, \vec{v}_h)} = k_h^*$ for the hth privilege level, it takes the following parameters $((v_1, \cdots, v_{\mu_1}), \cdots, (v_{\mu_{h-1}+1}, \cdots, v_{\mu_h}))$, pk and sk as inputs, and uniformly selects σ_j and η from $\mathbb{F}_\varphi$ for

$\forall j \in (1, \cdots, h)$, and runs the following formula:

$$k_h^* = \sum_{j=1}^{h} \sigma_j \Big(\sum_{j'=\mu_{j-1}+1}^{\mu_j} v_{j'} b_{j'}^* \Big) + \eta \cdot b_{z+1}^* + (1-\eta) \cdot b_{z+2}^* \tag{4}$$

(2) *Delegate*$^+$. The MOPSE$^+$-**Delegate** is the same as HPE-delegate in [31]. For more detail, please refer to [31].

(3) *CiphertextGen*$^+$ and *LabelGen*$^+$. As described in the Section IV-B.1, the finally outputs of the two sub-algorithms are $P_{i,d} = \mathcal{H}_c(f_i(C_{i,d}))$ and C_d, respectively.

(4) *Labelblind*$^+$. Given $P_{i,d}$ and C_d, data provider i with hth privilege chooses a random ζ from $\mathbb{F}_\varphi$, and generates the ciphertext $C_{h,d} = (C_1, C_2)$ according to the following formula.

$$\begin{aligned} C_1 &= \Big(\sum_{j=1}^{\ell} \delta_j \Big(\sum_{j'=\mu_{j-1}+1}^{\mu_j} x_{j'} b_{j'} \Big) + \zeta \cdot (b_{z+1} + b_{z+2}) \\ &\quad + \delta_{z+3} \cdot b_{z+3} \Big) \cdot C_d \\ C_2 &= g_{\mathcal{T}}^{\zeta} \end{aligned} \tag{5}$$

Note that, $\vec{X}_\ell$ is formed by concatenating $\ell - h$ random vectors $(\vec{x}_{h+1}^+, \cdots, \vec{x}_\ell^+)$ and $\vec{X}_h$, i.e., $\vec{X}_\ell$ is expressed as $(\vec{x}_1, \cdots, \vec{x}_h, \vec{x}_{h+1}^+, \cdots, \vec{x}_\ell^+)$. Finally, data provider concatenates the ciphertext $C_{h,d} = (C_1, C_2)$ and $P_{i,d}$, and returns the result $(C_1, C_2)||P_{i,d}$.

C. TRANSFORMING ENCRYPTED INDEXES UNDER MOPSE$^+$

As well as MOPSE, the cloud server also merges multiple encrypted indexes by comparing $P_{i,d}$ in MOPSE$^+$, and then separates the merged index into two indexes: S$^+$ index and H index. Besides, the cloud server utilizes the hash function $\mathcal{H}_s$ to obtain the final H index.

D. TRAPDOOR GENERATION UNDER MOPSE$^+$

In view of MOPSE$^+$ scheme, we conclude that data owner generates the trapdoor $\widetilde{Q}$ with identical method described in Sec. III-F.

To generate trapdoors, data provider first encrypts the query condition q with the symmetric key K_s to obtain $\widehat{C}_q = \mathsf{AES}(K_s, q)$. Then, data provider takes $\widehat{C}_q$ and the secret key $sk_{(\vec{v}_1, \cdots, \vec{v}_{h'})}$ as inputs, and returns the trapdoor $\tilde{t}_{h',q}$ through the Eq. (6).

$$\begin{aligned} \tilde{t}_{h',q} &= (k_{h'}^* + \epsilon \cdot b_{z+1}^* + (-\epsilon) \cdot b_{z+2}^*) \cdot (\frac{1}{\widehat{C}_q}) \\ &= \Big(\sum_{j=1}^{h'} \sigma_j \cdot \Big(\sum_{j'=\mu_{j-1}+1}^{\mu_j} v_{j'} b_{j'}^* \Big) + (\eta + \epsilon) \cdot b_{z+1}^* \\ &\quad + (1 - \eta - \epsilon) b_{z+2}^* \Big) \cdot (\frac{1}{\widehat{C}_q}), \end{aligned} \tag{6}$$

where ϵ is random chosen from $\mathbb{F}_\varphi$, and h' means the privilege level. Likewise, if q denotes a range, we need to generate $\widehat{C}_q$ for each data in the range. To support multi-attribute query, we generate trapdoors for each attribute, and output the final trapdoor $\widetilde{T} = \{\tilde{t}_1, \cdots, \tilde{t}_\psi\}$ as well as that in MOPSE.

E. PRIVACY-PRESERVING QUERY UNDER MOPSE$^+$

Without loss of generality, we assume the cloud server receives a multi-attribute trapdoor, such as $\widetilde{Q}$ or $\widetilde{T}$. Then, the cloud server runs Algorithm 2, and returns the matched encrypted files to data owner or data providers.

$$e(C_1, \tilde{t}_{h',q}) = C_2 \tag{7}$$

Algorithm 2 *PrivacyQuery*$^+$

Input: (H, $\widetilde{Q}$) or (S$^+$, $\widetilde{T}$)
Output: Encrypted files

```
Extract 1st query attribute value in Q̃ or T̃ as q̃ or t̃;
Remove 1st query attribute value in Q̃ or T̃;
for Traverse each element χ in H/S+ index from left to right do
    if χ.P ∈ q̃ or χ.C satisfies with Eq. (7) then
        if χ belongs to a leaf node then
            return Encrypted file;
        else
            PrivacyQuery+(H.childnode, Q̃)
            or PrivacyQuery+(S+.childnode, T̃);
```

Next, we proof the correctness of Equation (7). Let the ciphertext $C_{h,d}$ and the trapdoor $\tilde{t}_{h',q}$ be the inputs. Since

$$C_1 = \Big(\sum_{j=1}^{\ell} \delta_j \cdot \Big(\sum_{j'=\mu_{j-1}}^{\mu_j} x_{j'} \cdot b_{j'} \Big) + \zeta \cdot (b_{z+1} + b_{z+2}) + \delta_{z+3} \cdot b_{z+3} \Big) \cdot C_d$$

and

$$\tilde{t}_{h',q} = \Big(\sum_{j=1}^{h'} \sigma_j \Big(\sum_{j'=\mu_{j-1}+1}^{\mu_j} v_{j'} b_{j'}^* \Big) + (\eta + \epsilon) \cdot b_{z+1}^* + (1 - \eta - \epsilon) \cdot b_{z+2}^* \Big) \cdot (\frac{1}{\widehat{C}_q}).$$

The proof is described as follows:

Proof:

1) If $h' > h$:

$$e(C_1, \tilde{t}_{h',q}) = g_{\mathcal{T}}^{\sum_{1 \le j \le h} \delta_j \cdot \sigma_j \cdot \vec{x}_j \cdot \vec{v}_j \cdot (\frac{C_d}{\widehat{C}_q})} \cdot g_{\mathcal{T}}^{\zeta \cdot (\frac{C_d}{\widehat{C}_q})} \cdot g_{\mathcal{T}}^{\sum_{h+1 \le j \le h'} \delta_j \cdot \sigma_j \cdot \vec{x}_j^+ \cdot \vec{v}_j \cdot (\frac{C_d}{\widehat{C}_q})} \tag{8}$$

$\because \forall j \in (h+1, h'), \vec{x}_j^+ \cdot \vec{v}_j \neq 0$
$\therefore$ Either $\widehat{C}_q = C_d$ or $\widehat{C}_q \neq C_d$

$$e(C_1, \tilde{t}_{h',q}) = g_{\mathcal{T}}^{(\sum_{h+1 \le j \le h'} \vec{x}_j^+ \cdot \vec{v}_j + \zeta) \cdot (\frac{C_d}{\widehat{C}_q})} \neq g_{\mathcal{T}}^{\zeta} = C_2 \tag{9}$$

2) If $h' \leq h$:

$$e(C_1, \tilde{t}_{h',q}) = g_{\mathcal{T}}^{\sum_{1\leq j\leq h'} \delta_j \cdot \sigma_j \cdot \vec{x}_j \cdot \vec{v}_j \cdot (\frac{C_d}{C_q})} \tag{10}$$

$\because \forall j \in (1, h'), \vec{x}_j \cdot \vec{v}_j = 0$

$\therefore$ If $\widehat{C}_q = C_d$,

$$e(C_1, \tilde{t}_{h',q}) = g_{\mathcal{T}}^{(\sum_{1\leq j\leq h'} \vec{x}_j^+ \cdot \vec{v}_j + \zeta) \cdot (\frac{C_d}{C_q})} = g_{\mathcal{T}}^{\zeta} = C_2 \tag{11}$$

If $\widehat{C}_q \neq C_d$,

$$e(C_1, \tilde{t}_{h',q}) = g_{\mathcal{T}}^{(\sum_{1\leq j\leq h'} \vec{x}_j^+ \cdot \vec{v}_j + \zeta) \cdot (\frac{C_d}{C_q})} \neq g_{\mathcal{T}}^{\zeta} = C_2 \tag{12}$$

□

V. PERFORMANCE ANALYSIS

A. INDEX GENERATION

Assume that there are ψ attributes and γ entities for each index. The arbitrary attribute value belongs to the range [min, max], and we define τ as max − min +1. Namely, τ denotes the max number of elements of an attribute. For an MDBT, the height is equivalent to the attributes number ψ, and the overhead of inserting a entity to a MDBT is $\mathcal{O}(\psi \cdot \tau)$. Hence, generating a index with γ entities consumes $\mathcal{O}(\psi \cdot \tau \cdot \gamma)$. In the case of index generation, MOPSE and MOPSE$^+$ only need to build plaintext MDBT, so the time complexity of MOPSE and MOPSE$^+$ are both $\mathcal{O}(\psi \cdot \tau \cdot \gamma)$.

B. INDEX ENCRYPTION

In MOPSE and MOPSE$^+$, we utilize AES-128 as the symmetric encryption method, but it is not our focus point. Hence, we here not discuss and list the complexity of AES-128 (for more information, please refer to [28]). During indexes encryption, each label consists of two categories: $M_{i,d}/C_{h,d}$ and $P_{i,d}$. For each $P_{i,d}$, MOPSE and MOPSE$^+$ are both taking $\mathcal{O}(1)$ to compute the mapping function f. For $M_{i,d}$, MOPSE needs $\mathcal{O}(w)$ hash and $\mathcal{O}(1)$ encryption. While, in MOPSE$^+$, the ciphertext C_1 is the result of the point multiplication between pk and the C_d, and the ciphertext C_2 is also a point multiplication result. Thus, the encryption time is $\mathcal{O}(z + 3)$.

C. INDEXES TRANSFORMING

In indexes transforming, it consists of two phases: indexes merging and index segmenting. For the former, it can be viewed as merging multiple MDBTs. For simplify, we consider the index merging of two MDBTs. Since the $P_{i,d}$ values of the same plaintext for different data providers are distinguish with a high possibility, the indexes merging can be considered as their roots merging. The maximum number of elements for a root is τ, thus the computation overhead of merging two roots is $\mathcal{O}(\tau^2)$. Hence, merging two indexes takes $\mathcal{O}(\tau^2)$. For the index segmenting, the time complexity is equivalent to that of traversing an merged MDBT, and the number of MDBTs is n. Thus, it takes $\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$.

TABLE 2. Time overhead analysis.

Working Phases	Methods	Time Complexity
Index Generation	MOPSE	$\mathcal{O}(\psi \cdot \tau \cdot \gamma)$
	MOPSE$^+$	$\mathcal{O}(\psi \cdot \tau \cdot \gamma)$
Index Encryption	MOPSE	$\mathcal{O}(w) + \mathcal{O}(1)$
	MOPSE$^+$	$\mathcal{O}(z + 3)$
Indexes Merging	MOPSE	$\mathcal{O}(\tau^2)$
	MOPSE$^+$	$\mathcal{O}(\tau^2)$
Index Segmenting	MOPSE	$\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$
	MOPSE$^+$	$\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$
Query for data providers	OPSE	$\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$
	MOPSE	$\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma \cdot (2w - 2))$
	MOPSE$^+$	$\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma \cdot (z + 3))$
Query for data owner	OPSE	$\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$
	MOPSE	$\mathcal{O}(\psi \cdot \tau \cdot \gamma)$
	MOPSE$^+$	$\mathcal{O}(\psi \cdot \tau \cdot \gamma)$

D. QUERY

In MEIM scheme, the query consists of two categories: queries from data providers and queries from data owner. The time complexity of a query on an MDBT is $\mathcal{O}(\psi \cdot \tau \cdot \gamma)$. Thus, the query time complexity for data providers with OPSE is $\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$. However, with MOPSE, multiple indexes are merged into one indexes. Besides, Chen and Liu [25] pointed that the number of prefixes in a query is at most $2w$-2. Thus, the time complexity for data providers' query under MOPSE is $\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma \cdot (2w - 2))$. In MOPSE$^+$, each matching takes z+3 pairing operations, so the computation overhead of a query is $\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma \cdot (z + 3))$. For data owner's query, it only needs to match $P_{i,d}$. With either MOPSE or MOPSE$^+$, the $P_{i,d}$ query on an merged H index are both $\mathcal{O}(\psi \cdot \tau \cdot \gamma)$. Nevertheless, with OPSE, data owners need to generate multiple queries, so the query for data owner is $\mathcal{O}(n \cdot \psi \cdot \tau \cdot \gamma)$.

VI. SECURITY ANALYSIS

A. INDEX PRIVACY

In MEIM mechanism, the index privacy contains two categories: the privacy for S/S$^+$ index and the privacy for H index.

In MOPSE, the indexes consist of S index and H index. For S index, in view of Section III-C, the cloud server only receives the secure hash values of $M_{i,d}$ converted from data items in the outsourcing phase. Without knowing the secure hashing and the keys, it is computationally infeasible to compute the actual values of the corresponding $M_{i,d}$ for the attacker. For the analysis of information leaking of $\text{HMAC}_{K_i}(\bullet)$, Chen and Liu [29] pointed that the cloud server need large steps to reveal data (for more details, please refer to [29]). Note that if the cloud server and data providers are both compromised, the cloud server may reveal encrypted data issued by data providers through brute-force attacks. In that case, the cloud server knows the secret key K_i for the HMAC_{K_i} function. Since the function has one-wayness property of HMAC_{K_i}, the cloud server cannot reveal d directly using $\text{HMAC}_{K_i}(d)$ and K_i. But the attacker may compute the HMAC_{K_i} results of the numericalized prefixes for all possible values in the data domain with a brute-force

manner, and compare the results with the collected data. By the comparison, the cloud server can reveal data providers' outsourced data. However, in practice, the cloud server need to take prohibitive computational cost for a large data domain. For H index, the cloud server obtains the $P_{i,d}$ for a value d. Without knowing the security key K_i, f_i and R_i, the cloud server cannot infer the actual value d according to $P_{i,d}$ even if it knows $\mathcal{H}_c$. Besides, due to without knowing R_i, the cloud server cannot directly infer the actual value according to $P_{i,d}$. Furthermore, we assume that the cloud server will not collude with data providers, thus data provider cannot infer the actual value d without knowing the hash function $\mathcal{H}_s$.

In MOPSE$^+$, the indexes includes S$^+$ and H index, respectively. For H index, due to identical generation with that in MOPSE, we can follow the security analysis in MOPSE. For S$^+$ index, we propose Theorem 2 to demonstrate our proposed method is selectively identification-hiding against CPA.

Proof: The proof is elaborated in Appendix X.

Theorem 2: Our proposed scheme is selectively identification-hiding against CPA under the RDSP and IDSP assumptions [30]. For any adversary $\mathcal{A}$, there exist probabilistic machines $\mathcal{B}_1$ and $\mathcal{B}_2$, whose running times are essentially the same as that of $\mathcal{A}$, such that for any security parameter λ,

$$Adv^{IH}_{\mathcal{A}}(\lambda) \leq Adv^{RDSP}_{\mathcal{B}_1}(\lambda) + Adv^{IDSP}_{\mathcal{B}_2}(\lambda) + 3v/\varphi, \quad (13)$$

where v is the number of adversary's queries.

B. QUERY PRIVACY

In MEIM mechanism, the query privacy is divided into two cases: the privacy for data providers and the privacy for data owner.

In MOPSE, the query of data providers is based on HMAC_{K_i}. Hence, we conclude that the security analysis of the query of data providers can follow that of the S index. The query of data owner is based on the range R, the hash functions $\mathcal{H}_c$ and $\mathcal{H}_s$. Without knowing the $\mathcal{H}_c$ and the range R, the cloud server cannot infer the actual query value q according to $\mathcal{H}_s$ and the encrypted query. Likewise, data providers cannot know the query value q due to without knowing the $\mathcal{H}_s$.

In MOPSE$^+$, the query of data providers is based on our proposed scheme and we prove the security in the security analysis of the index. Thus, we can make sure that the attacker cannot infer the actual value q of query and the hierarchical identification of data providers through encrypted data. For the query of data owner, since we utilize the same method like in MOPSE, and the security proof has been proved in the above paragraph. The attacker cannot infer the value q through the encrypted query of data owner.

C. AUTHORIZED SEARCH

In MOPSE, the attacker, without knowing the secret key of data providers, cannot generate fake queries of data providers. In common, the attacker can only obtain either the hash function $\mathcal{H}_c$ or $\mathcal{H}_s$, thus the attacker cannot directly generate a fake query of data owner. In MOPSE$^+$, the attacker, without knowing the identification secret of data providers, cannot generate fake queries of data providers. Likewise, the attacker also cannot generate a fake query of data owner. Thus, we conclude that MEIM achieves the goal of authorized search.

VII. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the MEIM mechanism by three schemes: OPSE, MOPSE and MOPSE$^+$. We both implement schemes in C++, and MOPSE$^+$ is with the Pairing-Based Cryptography (PBC) Library [32]. Note that the type A elliptic curve parameter is adopted, where the group order is 160-bits, providing 80-bit security strength equivalently. For experiment dataset, since no real PHR datasets are publicly available for academic purposes, we apply MEIM to Nursery Dataset obtained from the UCI Machine Learning Repository [21]. The dataset is used in the previous research on searchable encryption [3], [12]. Briefly, the dataset comprises 12,960 instances, which contain eight categories reaching up to five values each. These experiments are carried out on an IBM workstation running RedHat Linux with a Intel(R) Xeon(R) CPU E5606 (with four cores) @2.13GHz with 12GB of random access memory. The experimental results show that these algorithms on MEIM perform well.

In MOPSE, categories and values are deemed to be attributes and attribute values, respectively. Each value in Nursery Dataset will be coded into integers by Unicode [23]. However, in MOPSE$^+$, we follow the experimental setup in [30] to set privileges. Namely, each instance is randomly defined as a hth privilege, where the front h categories are utilized to generate the vectors of the secret/public keys for the corresponding h privilege, and the rest categories are viewed as the random vectors. Here, the category values are converted into elements in $\mathbb{F}_\varphi$ using SHA-1 hash algorithm. To evaluate the efficiency of our schemes for various instances' number, we divide the data set into ten subsets, and each contains 1296 instances. In the evaluation of the encryption and query efficiency, we utilize multiple subsets, from one to ten, to test the encryption and query time. Note that our dataset are organized by MDBT, and stored in-memory. The following experimental results are based on this premise.

A. INDEX GENERATION

To evaluate the performance of index generation, we utilize n subsets to generate n indexes with MDBT. Each MDBT contains eight layers corresponding to eight attributes, respectively.

According to the performance analysis of index generation, we obtains two conclusions: (1) the time complexity of MOPSE is equal to that of MOPSE$^+$; (2) the time complexity of generating a index is $\mathcal{O}(\psi \cdot \tau \cdot \gamma)$.

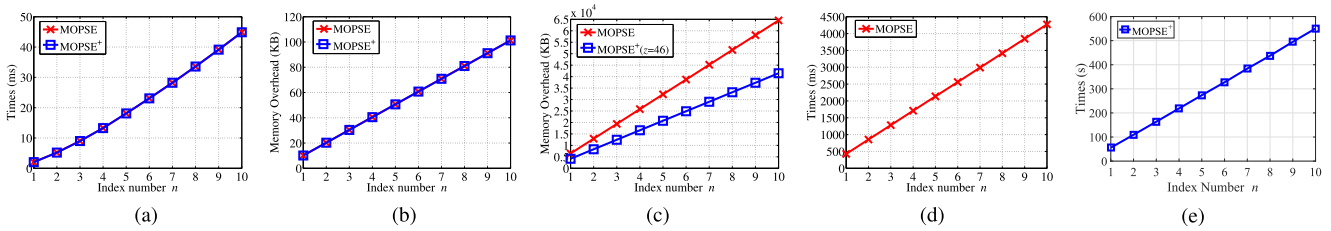

FIGURE 6. Fig. 6a and 6b show the index generation time and memory overhead, respectively. The memory overhead of encrypted indexes is represented in Fig. 6c. Fig. 6d and 6e depict the encrypting overhead under MOPSE and MOPSE$^+$. (a) Index generation time. (b) Index memory overhead. (c) En-index memory overhead. (d) Encryption under MOPSE. (e) Encryption under MOPSE$^+$.

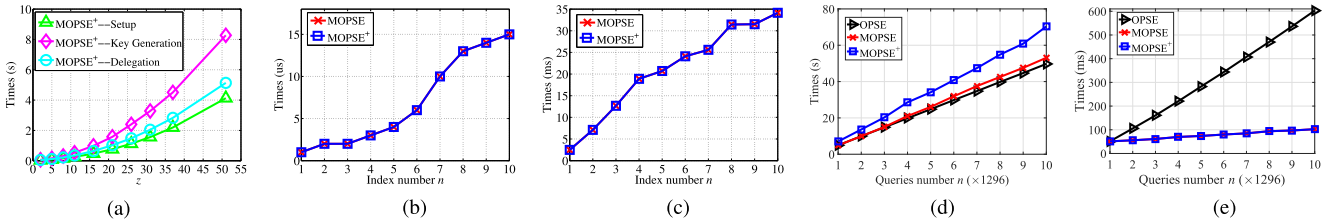

FIGURE 7. Fig. 7a shows the time of setup, key generation and delegation generating in MOPSE$^+$. The transforming overhead includes indexes merging time (Fig. 7b) and index segmenting time (Fig. 7c). Figure 7d and 7e depict the query consumption for data providers/owners under OPSE, MOPSE and MOPSE$^+$, respectively. (a) Time overhead in MOPSE$^+$. (b) Indexes merging time. (c) Index segmenting time. (d) Query of data providers. (e) Query of data owner.

Fig. 6a and 6b present the average running time and memory consumption for generating n indexes ($1\leq\ n\ \leq 10$). In our experiment, we set ψ as 8, thus the generation time of indexes increases with the multiple of τ and γ. As shown in Fig. 6a, when the index numbers pick up $\{3, 7, 9\}$, the time overhead are $\{8.845ms, 27.849ms, 39.080ms\}$. Meanwhile, the points of MOPSE and MOPSE$^+$ are overlap, which satisfies the conclusion (2). For memory usage of indexes, Fig. 6b shows that it is linear growth with an increasing number of n. When n is 2, the memory usage is 20.25KB.

B. INDEX ENCRYPTION

To evaluate the performance of index encryption, we take the above indexes as input, and runs MOPSE and MOPSE$^+$ schemes to output the encryption time.

In MOPSE, we utilize the AES scheme as a symmetric encryption, and $\mathrm{HMAC}_{K_i}(\bullet)$ to generate $M_{i,d}$. As shown in Figure 6d, the encryption overhead for indexes increases with the indexes number n. When n=3, the encryption time is about $1289.894ms$.

In MOPSE$^+$, we use our proposed method HPBPE to generate $C_{h,d}$. Figure 7a shows the time overheads for setup, key generation and delegation in MOPSE$^+$. In setup, the overhead includes $\mathcal{O}(z_0^2) = \mathcal{O}(z^2)$ exponentiations each (where z_0 is equivalent to z+3). When z is 31, the setup time is about $1.56s$. To evaluate the performance of key generation and delegation, we choose h ($1\leq\ h\ \leq 8$) identification from the identification universe in each identification to form a query. That is, the vector $\vec{v}$ does not have element $0 \in \mathbb{F}_\varphi$. According to Figure 7a, the direct key generation consumes relatively long time, while the delegation consumes less time. The reasons are that the former is processed by the central trusted authority because it is usually an one-time operation; while the latter is experienced by Level-2 local trusted authority and users under a Level-1 local trusted authority. Notice that the capability generation/delegation times both scale as $\mathcal{O}(z_0^2)$. Fig. 6e shows the encryption consumption with MOPSE$^+$, and we conclude that the encryption overhead in MOPSE is lower than that in MOPSE$^+$ through comparing Figure 6d and 6e. For a attribute value, MOPSE$^+$ takes $42.5ms$ to encrypt it.

For memory cost in MOPSE, the label for a attribute value consists of two parts: $M_{i,d}$ and $P_{i,d}$. Each prefix in $M_{i,d}$ is with 128-bit, and the $P_{i,d}$ is a integer value. While in MOPSE$^+$, the memory overhead of $C_{h,d}$ need $65(z_0+1)$B. For a attribute value, when z picks up 46 , the size is equal to merely 3.2KB. Figure 6c shows the comparison of the memory overhead between MOPSE and MOPSE$^+$.

C. ENCRYPTED INDEXES TRANSFORMING

To evaluate the overhead of the merging for multiple encrypted indexes, the cloud server merges the aforementioned encrypted indexes due to the comparison of the suffixes. Fig. 7b concludes that the merging time grows linearly with the elements number in the root which is reasonable because ciphertexts for the same plaintext on different indexes are unequal. When n is $\{5, 10\}$, the merging time is $\{4us, 15us\}$, respectively. According to the aforementioned analysis, ciphertexts for one plaintext are distinct, thus the merging cannot lead to a changeable in memory overhead. While for the segmenting, the consumption is a positive correlation with the total numbers of the nodes. The results are shown in Fig. 7c, e.g., n is 9, the segmenting time is $31ms$.

D. PRIVACY-PRESERVING QUERY

The performance of the privacy-preserving query is an key evaluation for the cloud in MEIM. To evaluate the query efficiency of data owner and data providers, we separately use $n\times1296$ ($1\leq n \leq10$) queries to test the average query time on the above indexes. Fig. 7d depicts the query overhead for queries issued by data providers with OPSE, MOPSE and MOPSE$^+$. As shown in Fig. 7d, we can conclude that the query performance in OPSE is greater than that in MOPSE and MOPSE$^+$, and the cloud server only takes 5.4*ms* averagely to process a query with eight attributes in MOPSE$^+$. Fig. 7e presents the query overhead for data owner, and the query overhead is the same between with MOPSE and with MOPSE$^+$. As shown in Figure 7e, the cloud server takes 102.51*ms* to process 12960 queries. Hence, the performance results demonstrate that MEIM is an efficient method for data owner querying.

VIII. RELATED WORK

Recently, privacy preserving in PHRs have drawn researchers' attention [2]–[7]. In this section, we review three categories of work: searchable encryption, order-preserving symmetric encryption and other related work. Search encryption schemes guarantee that the untrusted entity gains nothing about what data owners are searching for, and order-preserving symmetric encryption can guarantee the order of the ciphertexts following with that of the plaintexts. There has been a lot of works for searchable encryption [3], [12]–[19], [33], order-preserving symmetric encryption [20], [24], [33], [34], and other related work [41]–[46].

A. SEARCHABLE ENCRYPTION

With oursoucing encrypted data to the cloud, some researchers [3], [11]–[19], [33] proposed searchable encryption. In [3], Li *et al.* proposed a framework to address the problem of authorized private keyword searches (APKS) upon encrypted PHI. Song *et al.* [14] proposed symmetric key cryptography upon searchable encryption, and Boneh *et al.* [15] firstly proposed the public key cryptography on searchable encryption scheme. In [18], Liu *et al.* proposed EIRQ to support rank query, in which the queries with higher rank can retrieve higher percentage of matched files. Besides, Zhang *et al.* [33] focused on multi-keyword ranked search upon encrypted data. However, these schemes only support equal query, while not support range query. In [17], the authors constructed the index on the plaintexts and encrypted each page of the index separately. This scheme can reduce the communication for users' queries, but users need to manage amount of secret key. During the multi-source cloud personal health record, the low query efficiency and the highly communication overhead cause this scheme becomes undesirable. Among others, some researchers [25], [34], [36]–[38] focused on secure single- or multi- dimensional range queries. But the protocols [36]–[38] may return items that do not satisfy the query. Although SafeQ [25] can output the precise query results, the optimized version of SafeQ utilizes Bloom filter, thus leading to unprecise query results. To obtain the precise query results, Yi *et al.*, [34] proposed a protocol QuerySec to enable storage nodes to process queries correctly without leaking data and queries. Besides, Liu *et al.*, [35] proposed an MDS-SSE scheme to support multi-source scenario, but they ignored the hierarchical structure of data owners.

B. ORDER-PRESERVING SYMMETRIC ENCRYPTION

The order preserving encryption is utilized to encrypt the index to preserve the order of the ciphertexts with that of the plaintexts. Agrawal *et al.* [20] proposed an order preserving symmetric encryption (OPSE) scheme. Furthermore, Boldyreva *et al.* [24] proposed a modular order preserving encryption. Besides, Yi *et al.* [34] proposed an order preserving function to encode data in sensor network, and Zhang *et al.* [33] introduced identical function to implement the query on the relevance scores of keywords on the cloud. However, these schemes are only suitable for single source scenario because the numeric order of various indexes cannot be preserved with various OPSE. Besides, Liang *et al.* [40] proposed an MPOPE to address multi-provider problem in cloud. However, they ignored a practical problem, i.e., hierarchical authentication query.

C. OTHER RELATED WORK

Data access control is another vital issue in cloud storage system. Many researchers [41]–[46] focused on multi-authority access control scenario. In [41], Yang *et al.* aimed to address multi-authority problem, and proposed a data access scheme DAC-MACS. However, Hong *et al.* [42] found that Yang's work may be attacked due to utilizing a bidirectional re-encryption method, and described their attack method. To address the data privacy problem and the user identity privacy both, Jung *et al.* [43] proposed AnonyControl scheme. In [44], Xue *et al.* stated that the prior work cannot overwhelm the common shortcoming, like low efficiency and single-point bottleneck. So they proposed RAAC, a robust access control scheme. To reduce the overhead of decryption, Chase and Chow [45] proposed a scheme to remove the trusted central authority. In [46], Li *et al.* proposed a threshold multi-authority ciphertext-policy attribute-based encryption access control solution (named TMACS). However, none of them can be directly utilized to address our problem due to ignoring hierarchical authentication query problem.

IX. CONCLUSION AND DISCUSSION

In this paper, we explore the problem of privacy-preserving query for multi-source in the cloud-based PHR environment. Different from prior works, our proposed MEIM mechanism enables authenticated data owner to achieve secure, convenient, and efficient query over multiple data providers' data. To implement the efficient query, we introduce MDBT as the data structure. To reduce the overhead of query generation of data owner, and allow the cloud server to

securely query, we propose a novel multiple order-preserving symmetric encryption (MOPSE) scheme. To make our model more practical, we propose an enhanced multiple order-preserving symmetric encryption (MOPSE$^+$) scheme to satisfy the hierarchical authenticated query. Moreover, we leverage rigorous security proof to prove that our schemes are security. Finally, we demonstrate that the MEIM mechanism is computationally efficient by implementing our schemes and running in a real dataset.

Alghought our work only focuses on CB-PHR system, it can be theoretically extended to various scenarios, mobile data collection, recommendation system, and so forth. However, the devices, like mobile devices, have limited computation and memory resource. For all this, we will discuss light-weight schemes in our furture work.

X. PROOF OF SELECTIVELY IDENTIFICATION-HIDING AGAINST CPA

To prove the security of our proposed scheme, the following five games derived from [31] are deployed, i.e., *Game* 0 to *Game* 4. *Game* 0 is the original selectively identification-hiding game. *Game* 1 is an extension of the concept of *Game* 0. The **Setup**$^+$ (instead of **Delegate**$^+$) algorithm in *Game* 2 presents the query of a delegated key. *Game* 3 means that the plaintext part of the target ciphertext and trapdoor is randomized. While the randomly of the ciphertext part of the target ciphertext and trapdoor is in *Game* 4.

- *Game* 0: Primitive game (Defined as [31]).
- *Game* 1: Apart from the following condition, *Game* 1 is defined as the same with *Game* 1 in [31]. The adversary $\mathcal{A}$ gives the challenge plaintexts $C_d^{(0)}$ and $C_d^{(1)}$ to challenger $\mathcal{C}$, who computes the ciphertext (C_1, C_2) and the trapdoor $\tilde{t}_{h',d}$ as follows and returns it to $\mathcal{A}$.

$$C_1 = (\sum_{j=1}^{z} x_j^+ b_j + \zeta(b_{z+1} + b_{z+2}) + \delta_{z+3} b_{z+3}) \cdot C_d$$
$$C_2 = g_{\mathcal{T}}^{\zeta} \tag{14}$$
$$\tilde{t}_{h',d} = (\sum_{j=1}^{h'} v_j^+ b_j^* + (\eta + \epsilon) \cdot b_{z+1}^* + (1 - \eta - \epsilon) \cdot b_{z+2}^*) \cdot (\frac{1}{C_d}), \tag{15}$$

  where $\zeta, \delta_{z+3}, \eta$ and $\epsilon \xleftarrow{U} \mathbb{F}_\varphi$.
- *Game* 2: Because **Setup**$^+$ and **Delegate**$^+$ algorithms are equivalent to that in [31], the definition of *Game* 2 is the same with that of [31].
- *Game* 3: Except the ciphertext $C_{z,d}$ consisting of C_1 and C_2 and the trapdoor $\tilde{t}_{h',d}$ are issued by the following formulas, *Game* 3 is the same with *Game* 2.

$$C_1 = (\sum_{j=1}^{z} x_j^+ b_j + \zeta_1 b_{z+1} + \zeta_2 b_{z+2} + \delta_{n+3} b_{n+3}) \cdot C_d$$
$$C_2 = g_{\mathcal{T}}^{\zeta} \tag{16}$$
$$\tilde{t}_{h',d} = (\sum_{j=1}^{h'} v_j^+ b_j^* + (\eta + \epsilon_1) \cdot b_{z+1}^* + (1 - \eta - \epsilon_2) \cdot b_{z+2}^*) \cdot (\frac{1}{C_d}), \tag{17}$$

  where $\zeta, \zeta_1, \zeta_2, \delta_{z+3}, \epsilon_1, \epsilon_2, \eta \xleftarrow{U} \mathbb{F}_\varphi$.
- *Game* 4: Except the ciphertext C_1 and C_2 is issued by the following formula, it is indistinctive between *Game* 4 and *Game* 3.

$$C_1 = (\sum_{t=1}^{h'} u_j b_j + \zeta_1 b_{z+1} + \zeta_2 b_{z+2} + \delta_{z+3} b_{z+3}) \cdot C_d$$
$$C_2 = g_{\mathcal{T}}^{\zeta} \tag{18}$$
$$\tilde{t}_{h',d} = (\sum_{j=1}^{h'} \omega_j^+ b_j^* + (\eta + \epsilon_1) \cdot b_{z+1}^* + (1 - \eta - \epsilon_2) \cdot b_{z+2}^*) \cdot (\frac{1}{C_d}), \tag{19}$$

  where $\zeta, \zeta_1, \zeta_2, \delta_{z+3}, \epsilon_1, \epsilon_2 \xleftarrow{U} \mathbb{F}_\varphi$ and $\vec{u}=(u_1, \cdots, u_z)$, $\vec{\omega}=(\omega_1, \cdots, \omega_z) \xleftarrow{U} \mathbb{F}_\varphi^z \setminus \{\vec{0}\}$.

We assume that $Adv_{\mathcal{A}}^{IH}$ for *Game* 0 is $Adv_{\mathcal{A}}^{(0)}$, and the advantage of $\mathcal{A}$ for *Game* j corresponds to $Adv_{\mathcal{A}}^{(j)}$, where $j \in [1, 4]$. Since *Game* 1 is an extension of the concept of *Game* 0, $Adv_{\mathcal{A}}^{(0)}(\lambda)$ is the same with $Adv_{\mathcal{A}}^{(1)}(\lambda)$. Besides, $Adv_{\mathcal{A}}^{(4)}$ is equal to 0 by the in [31, Lemma 4].

According to the analysis of evaluating the gaps between pairs of $Adv_{\mathcal{A}}^{(j)}(\lambda)$. We conclude that $Adv_{\mathcal{A}}^{IH}(\lambda) = Adv_{\mathcal{A}}^{(0)}(\lambda) = Adv_{\mathcal{A}}^{(1)}(\lambda) \leq \sum_{j=1}^{3} |Adv_{\mathcal{A}}^{(j)}(\lambda) - Adv_{\mathcal{A}}^{(j+1)}(\lambda)| + Adv_{\mathcal{A}}^{(4)}(\lambda) \leq Adv_{\mathcal{A}}^{RDSP}(\lambda) + Adv_{\mathcal{A}}^{IDSP}(\lambda) + 3v/\varphi$.

ACKNOWLEDGMENT

This paper was presented in the Proceedings of the IEEE Symposium on Computers and Communications (ISCC'15) [39].

REFERENCES

[1] C. Wang, B. Zhang, K. Ren, J. M. Roveda, C. W. Chen, and Z. Xu, "A privacy-aware cloud-assisted healthcare monitoring system via compressive sensing," in *Proc. INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 2130–2138.

[2] J. Sun, X. Zhu, C. Zhang, and Y. Fang, "HCPP: Cryptography based secure EHR system for patient privacy and emergency healthcare," in *Proc. ICDCS*, Minneapolis, MN, USA, Jun. 2011, pp. 373–382.

[3] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *Proc. ICDCS*, Minneapolis, MN, USA, Jun. 2011, pp. 383–392.

[4] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient controlled encryption: Ensuring privacy of electronic medical records," in *Proc. ACM Workshop CCS*, New York, NY, USA, 2009, pp. 103–114.

[5] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, Jan. 2013.

[6] M. Li, S. Yu, K. Ren, and W. Lou, "Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings," in *Proc. SecureComm*, Singapore, Sep. 2010, pp. 89–106.

[7] X. Ma, Y. Zhu, and X. Li, "An efficient and secure ridge regression outsourcing scheme in wearable devices," *Comput. Elect. Eng.*, vol. 63, pp. 246–256, Oct. 2017, doi: 10.1016/j.compeleceng.2017.07.019.
[8] J. Liu, X. Huang, and J. K. Liu, "Secure sharing of personal health records in cloud computing: Ciphertext-policy attribute-based signcryption," *Future Generat. Comput. Syst.*, vol. 52, pp. 67–76, Nov. 2015.
[9] P. Scheuermann and M. Ouksel, "Multidimensional B-trees for associative searching in database systems," *Inf. Syst.*, vol. 7, no. 2, pp. 123–137, 1982.
[10] K. Xue, J. Hong, Y. Xue, D. S. Wei, N. Yu, and P. Hong, "CABE: A new comparable attribute-based encryption construction with 0-encoding and 1-encoding," *IEEE Trans Comput.*, vol. 66, no. 9, pp. 1491–1503, Sep. 2017.
[11] K. Xue, S. Li, J. Hong, Y. Xue, N. Yu, and P. Hong, "Two-cloud secure database for numeric-related SQL range queries with privacy preserving," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1596–1608, Jul. 2017.
[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. CCS*, Alexandria, VA, USA, 2006, pp. 1–28.
[13] Y. Zhu, Z. Huang, and T. Takagi, "Secure and controllable k-NN query over encrypted cloud data with key confidentiality," *J. Parallel Distrib. Comput.*, vol. 89, pp. 1–12, Mar. 2016.
[14] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE S&P*, Berkeley, CA, USA, May 2000, pp. 44–55.
[15] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. EUROCRYP*, Interlaken, Switzerland, 2004, pp. 506–522.
[16] Y. Zhu, Z. Wang, and Y. Zhang, "Secure k-NN query on encrypted cloud data with limited key-disclosure and offline data owner," in *Proc. PAKDD*, Auckland, New Zealand, 2016, pp. 401–414.
[17] B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu, "A framework for efficient storage security in RDBMS," in *Proc. EDBT*, Heraklion, Greece, 2004, pp. 147–164.
[18] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Efficient information retrieval for ranked queries in cost-effective cloud environments," in *Proc. INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 2581–2585.
[19] Y. Zhu, Z. Wang, and J. Wang, "Collusion-resisting secure nearest neighbor query over encrypted data in cloud, revisited," in *Proc. IWQoS*, Beijing, China, Jun. 2016, pp. 1–6.
[20] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. SIGMOD*, New York, NY, USA, 2004, pp. 563–574.
[21] A. Asuncion and D. Newman, "UCI machine learning repository," Tech. Rep., 2010.
[22] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Cooperative private searching in clouds," *J. Parallel Distrib. Comput.*, vol. 72, no. 8, pp. 1019–1031, 2012.
[23] *The Unicode Standard Version 2.0*, Univ. Consortium, Sep. 1997.
[24] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. CRYPTO*, Santa Barbara, CA, USA, 2011, pp. 578–595.
[25] F. Chen and A. Liu, "SafeQ: Secure and efficient query processing in sensor networks," in *Proc. INFOCOM*, San Diego, CA, USA, Mar. 2010, pp. 1–9.
[26] Y.-K. Chang, "Fast binary and multiway prefix searches for packet forwarding," *Comput. Netw.*, vol. 51, no. 3, pp. 588–605, Feb. 2007.
[27] *Order-Preserving Minimal Perfect Hashing*. [Online]. Available: http://www.nist.gov/dads/HTML/orderPreservMinPerfectHash.html
[28] "Advanced encryption standard (AES)," *Fed. Inf. Process. Standards Pub.*, vol. 197, pp. 311–441, 2001.
[29] F. Chen and A. X. Liu, "Privacy-and integrity-preserving range queries in sensor networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1774–1787, Dec. 2012.
[30] X. Yao, Y. Lin, Q. Liu, and Y. Zhang, "A secure hierarchical deduplication system in cloud storage," in *Proc. IWQoS*, Beijing, China, 2016, pp. 1–10.
[31] T. Okamoto and K. Takashima, "Hierarchical predicate encryption for inner-products," in *Proc. ASIACRYPT*, Tokyo, Japan, 2009, pp. 214–231.
[32] B. Lynn. *The PBC Library*. [Online]. Available: http://crypto.stanford.edu/pbc/
[33] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1566–1577, May 2016.
[34] Y. Yi, R. Li, F. Chen, A. X. Liu, and Y. Lin, "A digital watermarking approach to secure and precise range query processing in sensor networks," in *Proc. INFOCOM*, Turin, Italy, Apr. 2013, pp. 1950–1958.
[35] C. Liu, L. Zhu, and J. Chen, "Efficient searchable symmetric encryption for storing multiple source data on cloud," in *Proc. Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015, pp. 451–458.
[36] B. Sheng and Q. Li, "Verifiable privacy-preserving range query in two-tiered sensor networks," in *Proc. INFOCOM*, Phoenix, AZ, USA, Apr. 2008, pp. 46–50.
[37] J. Shi, R. Zhang, and Y. Zhang, "Secure range queries in tiered sensor networks," in *Proc. INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 945–953.
[38] R. Zhang, J. Shi, and Y. Zhang, "Secure multidimensional range queries in sensor networks," in *Proc. MobiHoc*, New Orleans, LA, USA, 2009, pp. 197–206.
[39] X. Yao, Y. Lin, Q. Liu, and S. Long, "Efficient and privacy-preserving search in multi-source personal health record clouds," in *Proc. ISCC*, Larnaca, Cyprus, Jul. 2015, pp. 803–808.
[40] J. Liang, Z. Qin, S. Xiao, J. Zhang, H. Yin, and K. Li, "MPOPE: Multi-provider order-preserving encryption for cloud data privacy," in *Proc. SecureComm*, Niagara Falls, ON, Canada, 2017.
[41] K. Yang, X. Jia, K. Ren, B. Zhang, and R. Xie, "DAC-MACS: Effective data access control for multiauthority cloud storage systems," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1790–1801, Nov. 2013.
[42] J. Hong, K. Xue, and W. Li, "Security analysis of attribute revocation in multi-authoritydata access control for cloud storage system," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 6, pp. 1315–1317, Jun. 2015.
[43] T. Jung, X.-Y. Li, Z. Wan, and M. Wan, "Privacy preserving cloud data access with multi-authorities," in *Proc. INFOCOM*, Turin, Italy, Apr. 2013, pp. 2625–2633.
[44] K. Xue *et al.*, "RAAC: Robust and auditable access control with multiple attribute authorities for public cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 953–967, Apr. 2017.
[45] M. Chase and S. S. M. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proc. CCS*, Chicago, IL, USA, 2009, pp. 121–130.
[46] W. Li, K. Xue, Y. Xue, and J. Hong, "TMACS: A robust and verifiable threshold multi-authority access control system in public cloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1484–1496, May 2016.

XIN YAO received the B.S. degree in computer science from Xidian University, China, in 2011, and the M.S. degree in software engineering, Hunan University, in 2013, where he is currently pursuing the Ph.D. degree with the College of Computer Science and Electronic Engineering. He is currently a Visiting Student with Arizona State University, Tempe, AZ, USA. His research interests include security and privacy issues in social network, cloud, and big data.

YAPING LIN received the B.S. degree in computer application from Hunan University, China, in 1982, and the M.S. degree in computer application from the National University of Defense Technology, in 1985, and the Ph.D. degree in control theory and application from Hunan University, in 2000. He has been a Professor and a Ph.D. supervisor with Hunan University since 1996. From 2004 to 2005, he was a Visiting Researcher with The University of Texas at Arlington. His research interests include machine learning, network security, and wireless sensor networks.

QIN LIU received the B.S. degree in computer science from Hunan Normal University, China, in 2004, and the M.Sc. degree in computer science from Central South University, in 2007, where he received the Ph.D. degree from the School of Information Science and Engineering. She is currently an Assistant Professor with Hunan University, China. Her research interests include security and privacy issues in cloud computing.

JUNWEI ZHANG received the B.S. and Ph.D. degrees in computer science from Xidian University, in 2004 and 2010, respectively. He is currently an Associate Professor with the School of Cyber Engineering, Xidian University. His primary research interests are cryptography and information security. He is a member of CCF and CACR.