Authorized Private Keyword Search over Encrypted Data in Cloud Computing

Ming Li*, Shucheng Yu†, Ning Cao* and Wenjing Lou*
*Dept. of ECE, Worcester Polytechnic Institute, email: {mingli,ncao,wjlou}@ece.wpi.edu
†Dept. of CS, University of Arkansas at Little Rock, email: sxyu1@ualr.edu

***Abstract*—In cloud computing, clients usually outsource their data to the cloud storage servers to reduce the management costs. While those data may contain sensitive personal information, the cloud servers cannot be fully trusted in protecting them. Encryption is a promising way to protect the confidentiality of the outsourced data, but it also introduces much difficulty to performing effective searches over encrypted information. Most existing works do not support efficient searches with complex query conditions, and care needs to be taken when using them because of the potential privacy leakages about the data owners to the data users or the cloud server. In this paper, using online Personal Health Record (PHR) as a case study, we first show the necessity of search capability authorization that reduces the privacy exposure resulting from the search results, and establish a scalable framework for Authorized Private Keyword Search (APKS) over encrypted cloud data. We then propose two novel solutions for APKS based on a recent cryptographic primitive, Hierarchical Predicate Encryption (HPE). Our solutions enable efficient multi-dimensional keyword searches with range query, allow delegation and revocation of search capabilities. Moreover, we enhance the query privacy which hides users' query keywords against the server. We implement our scheme on a modern workstation, and experimental results demonstrate its suitability for practical usage.**

I. Introduction

In recent years, cloud computing is gaining much momentum in the IT industry. Especially, we have seen the dramatic growth of public clouds, in which the computing resources can be accessed by the general public. One of the biggest advantages of a public cloud is its virtually unlimited data storage capabilities and elastic resource provisioning [3]. Many IT enterprizes and individuals are outsourcing their databases to the cloud servers, in order to enjoy the much lower data management cost than maintaining their own data centers. It has never been easier than now that a variety of users/clients could access or share information stored in the cloud, independent of their locations.

Despite enthusiasm around the cloud data service outsourcing model, its promises cannot be fulfilled until we address the serious security and privacy concerns that data owners have. The outsourced data may contain very sensitive information, such as Personal Health Records (PHRs), facebook photos, and business documents. Many people remain dubious about the levels of privacy protection of their data when stored in a server owned by a third-party cloud service provider. Given that there have been numerous reported data breaches related to cloud servers [2], which could be due to malicious attacks, theft or internal software bugs and errors, it can be claimed that the servers are not fully trustworthy. This implies that there is no absolute governance about how the owners' information will be used and whether the owners actually control access to their data. To cope with the tough trust issues and to ensure owners' control over their own privacy, applying data encryption on the documents before outsourcing has been proposed as a promising solution, which is already adopted by many recent works [26], [7], [24]. In this paper, we focus on the "multi-owner" setting, where the encrypted data are contributed by multiple owners and can be searched by multiple users.

With encrypted data, one of the key functionalities of a database system —- keyword search becomes an especially challenging issue. We will take PHR as the main motivating example. First we need to support frequently used complex query conditions efficiently. For example, a user may want to find out other patients with the same disease and symptoms from an encrypted PHR database, by submitting a query like "(20<*age*<30) **AND** (*sex*="female") **AND** (*illness*="diabetes")". Also, a medical researcher may query the database using the following: (*age*>50) **AND** (*region*="Massachusetts") **AND** (*illness*="cancer"). This class of boolean formulas feature conjunctions among different keyword fields and we will refer to them as *multi-dimensional keyword search* in this paper. To hide the query keywords from the server, it is apparently inefficient for a user to download the whole database and try to decrypt the records one by one. *Searchable encryption* (SE) has been proposed as a better solution [35], [18], [10], [15], [36], [13]; informally speaking, a user submits a "capability" encoding her query to the server, who searches through the encrypted keyword indexes created by the owners, and returns a subset of encrypted documents that satisfy the underlying query without ever knowing the keywords in the query and the index.

However, most existing solutions of SE lack the above query flexibility or do not bear enough efficiency. Early works mostly only support single-keyword search [18], [14], [15], [37]. Later, several multi-keyword search schemes were proposed [19], [5], [11], [33], [21], [32], [22], [34], [13] that enable conjunctive or disjunctive search formulas. It usually incurs high computational complexity to realize multi-dimensional range query over encrypted data due to the heavy reliance on public-key cryptography (PKC). There are only a few works that specifically tackle this challenging problem, such as [33]. In this paper, we also aim at supporting efficient multi-dimensional equality, subset and range queries.

On the other hand, in many existing SE schemes, the

legitimate users are often given a secret/private key that endows her unlimited capability to generate any query of her choice, which is essentially a "0" or "1" authorization. However, we note that "*fine-grained search authorization*" is an indispensable component for a secure data outsourcing system. Although the accesses to actual documents can be controlled by separate cryptographic enforced access control techniques such as attribute-based encryption [9], [39], [26], "0-1" search authorization may still lead to leakage of data owners' sensitive information. For example, if Alice is the only patient with a rare disease in a PHR database, by designing the query in a clever way (e.g., submitting two queries with/without the name of that disease and with Alice's demographic info), from the results a user Bob will be certain that Alice has that disease. Thus, we argue that a user should only be allowed to search for some specific sets of keywords; in particular, the authorization shall be based on a user's *attributes*. For instance, in a patient matching application in health social networks [28], [23], a patient should only be matched to patients having similar symptoms as her, while shall not learn any information about those who do not.

Furthermore, system scalability is an important concern for SE. For symmetric-key based SE schemes, the encryption and search capabilities are not separable, so a multi-owner system would require every owner to act as a capability distribution center, which is not scalable. PKC-based schemes do not have this problem, but if every user obtains restricted search capabilities from a central trusted authority (TA) who assumes the responsibility of authorization at the same time, it shall be always online, dealing with large workload, and facing the threat of single-point-of-failure. In addition, since the global TA does not directly possess the necessary information to check the attributes of users from different local domains, additional infrastructure needs to be employed (such as using a credential chain [27]). It is therefore desirable for the users to be authorized locally.

In this paper, we systematically study the problem of authorized private keyword searches (APKS) over encrypted data in cloud computing. We make the following main contributions. First, we propose a fine-grained authorization framework in which every user obtain search capabilities under the authorization of local trusted authorities (LTAs), based on checking for user's *attributes*. Part of the authorization of a higher level LTA is delegated to its lower-level LTAs. The central TA's task is reduced to minimum, and can remain semi-offline after initialization. Thus, our framework enjoys a high level of system scalability.

Under the above framework, we propose two solutions for searching on encrypted data, namely APKS and APKS$^+$. We make novel use of a recent cryptographic primitive, hierarchical predicate encryption (HPE), which features delegation of search capabilities. Both of our solutions enable efficient multi-dimensional queries with equality, subset and a class of simple range queries. Since the PKC-based SE schemes suffers from a type of dictionary attack that reveals the underlying keywords in a query to the server, in APKS$^+$ we enhance the query privacy by preventing that kind of attack with the help of additional proxy servers. To the best of our knowledge, the APKS$^+$ scheme is the first to achieve efficient multi-dimensional range query, capability delegation and query privacy simultaneously.

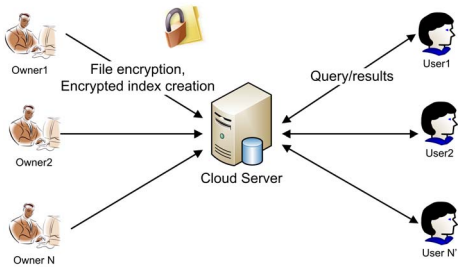


Fig. 1. System model for multi-owner data outsourcing in cloud computing.

Finally, we implement our scheme on a modern workstation and carry out extensive performance evaluation. Through experimental results we demonstrate that our scheme is suitable for a wide range of delay-tolerant database search applications.

II. Problem Formulation

A. System Model

We consider a cloud computing environment that hosts an outsourced database, based on which data sharing applications can be built. For illustration purposes, we will use an online PHR service (e.g., GoogleHealth [1]) as case study in this paper. The entities in the system are: data owners/users, trusted authorities, and the cloud server. In this paper, data *owner* refers someone who owns the information, e.g., a patient who encrypts her PHR data and wants them to be stored in the cloud server while preserving her privacy. The cloud server stores the encrypted data contributed by *multiple owners* in a database and performs search for the users. The "*users*" generally refer to those who can perform searches over the encrypted database. They could originate from various avenues, and usually need to search and access the data due to their professional responsibilities. We assume that the data contents are protected using separate, existing data encryption schemes [26], which is not the focus of this paper. The system architecture is illustrated in Fig. 1.

B. Threat Model

We assume the cloud server to be "honest-but-curious", which is also adopted by many existing works on SE [15], [36] and data security in cloud computing environments [7], [39], [26], [13]. That is, the server is "curious" to learn and infer the data contents or searchable index, but will honestly follow the protocol run. The server could also collude with any number of users to derive additional information about other users' queries and the encrypted data.

C. Design Goals

The system design of APKS over encrypted data in cloud computing should achieve the following main security and performance goals.

- **Index and Query Privacy**: The primary security goal is to prevent the cloud server from learning any useful

Field notation:	Z_1	Z_2	Z_3	Z_4	Z_5
Field name:	Age	Sex	Region	Illness	Provider
Alice:	25	Female	Worcester	Flu	Hospital A
Bob:	61	Male	Boston	Diabetes	Hospital B

TABLE I
ILLUSTRATION OF DATA STRUCTURE: (PLAINTEXT) KEYWORD INDEXES FOR PHRS OF TWO OWNERS.

information about the encrypted documents, indexes, and the users' queries, except what can be derived from the search results. Index privacy refers to confidentiality of the index, while query privacy protects users' queries.

- **Fine-grained Search Authorization, and Revocation**: It is equally important to prevent curious users from gaining additional information from the database than what they need to know. To reduce the risk of privacy exposure by unrestricted query capabilities, the users' search requests should be authorized in a fine-grained manner. In addition, there should exist a mechanism to revoke the search capability of a user.
- **Multi-dimensional Keyword Search**: The system should support multi-dimensional keyword search functionality, namely, we want to support conjunctions among different dimensions where in each dimension there can be multiple keywords (including equality, subset and range queries).
- **Scalability and Efficiency**: The system should allow multiple owners to encrypt and contribute data, while enabling a large number of users to search over multiple owners' data. In achieving this, the system should have high scalability, i.e., low key management overhead. Also, efficiency should be acceptable for per-search operation from a user's point of view.

In addition, other important system-level security requirements such as user authentication, access control and accountability can be realized by existing techniques [26], [39], [38], thus are not the main focus in this paper.

D. Notations and Definitions

Without special notice, we will use upper-case letters to denote variables and lower-case letters for values of variables.

Attributes and Keywords. An *attribute* generally refers to a category of property of an owner and her data, such as "age", "illness", and "provider". These attributes are *multi-valued* which can be either numerical or non-numerical, for example, "age" can have values as the integers 0-100, while "illness" has values as the names of every illness. Each attribute value is mapped to an element in $\mathbb{F}_q$, a finite field using a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{F}_q$. In the following, an attribute is also called as a "*dimension*" or "*field*" in the index, and each attribute value is a "*keyword*".

Data Structure. We assume each owner's original data consists of two parts: one is the actual file(s), $\mathcal{D}$, which may be encrypted using separate techniques (e.g., [7], [26]), and another is a searchable index $\vec{Z}$ for $\mathcal{D}$ consisting of m *dimensions*, and each dimension contains one *keyword*: $\vec{Z} := (z_1, z_2, \cdots, z_m)$. The collection of all owners' indexes can be regarded as a table of m columns, and an example is shown in Table I.

m, m'	Number of fields in original/converted index
d, d_i	Maximum # of ORs in a query over each/i-th dimension
k	Height of attribute hierarchy/expansion factor
$\vec{Z}, Z_i, z_i$	An index, the i-th index field and its value (keyword)
$Q, \widehat{Q}, w_i$	A query, its CNF form and a query keyword
$\vec{x}$	Plaintext vector
$\vec{v}$	Predicate vector
n	Length of plaintext/predicate vectors
ψ, ϕ	Mappings from $\vec{Z}$ to $\vec{x}$, and from $\widehat{Q}$ to $\vec{v}$

TABLE II
MAIN NOTATIONS.

Query. There are three types of basic query terms defined over a single dimension: equality ($Z_i = w_i$), subset ($Z_i \in \mathcal{S}_i$), and range query ($s \leq Z_i \leq t$). We consider *multi-dimensional keyword queries* which are formulas connecting query terms over different dimensions using **AND** ($\wedge$). For example,

$Q := (30 \leq Z_1 \leq 60) \wedge (Z_2 =$ "*") $\wedge$ ($Z_3 \in$ {"Boston", "Worcester"}) $\wedge$ ($Z_4 =$ "Diabetes").

The above can be converted into conjunctive normal form (CNF), where the subset and range queries are expressed using the equality and **OR** ($\vee$) operators. We denote the CNF form of a query Q as $\widehat{Q}$.

Predicates. We consider a class of predicates to be functions $\mathcal{F} := \{f_{\vec{v}} | \vec{v} \in \mathbb{F}_q^n\}$, where $f_{\vec{v}}(\vec{x}) = 1$ *iff.* $\vec{x} \cdot \vec{v} = 0$ [21]. We define a mapping ϕ from a query $\widehat{Q}$ to a predicate vector $\vec{v}$, and another, ψ from an index $\vec{Z}$ to a plaintext vector $\vec{x}$.

Search Capabilities. In searchable encryption, the user needs to submit a search capability (also called trapdoor) $T_{f_{\vec{v}}}$ encoding the predicate $f_{\vec{v}}$, with which the server evaluates $f_{\vec{v}}(\cdot)$ against the encrypted indexes. To define search delegation, we say that a capability $T_{f_{\vec{v}_1}}$ is more *restrictive* than another $T_{f_{\vec{v}_2}}$, if $\mathcal{Z}_1 \subseteq \mathcal{Z}_2$, where $\mathcal{Z}_1 := \{\vec{Z} = \psi^{-1}(\vec{x}) | f_{\vec{v}_1}(\vec{x}) = 1\}$, and $\mathcal{Z}_2 := \{\vec{Z} = \psi^{-1}(\vec{x}) | f_{\vec{v}_2}(\vec{x}) = 1\}$, and is denoted as $T_{f_{\vec{v}_1}} \prec T_{f_{\vec{v}_2}}$. For convenience, a capability is also denoted as T_Q for underlying query Q.

In addition, we denote "choose a value r uniformly at random from $\mathbb{F}_q$ by $r \in_R \mathbb{F}_q$". There are five algorithms: Setup, GenIndex, GenCap, Search and DelegateCap; their definitions are shown in our technical report [25] due to space limitations. Table II shows other main notations.

III. FINE-GRAINED AUTHORIZATION FRAMEWORK FOR APKS OVER ENCRYPTED DATA

The search authorization framework in this section adds another layer of fine-grained privacy protection beyond the underlying cryptographic mechanisms used for encrypted search or data access control. It is complementary and compatible with the "patient-centric" data access control framework in our previous work [26].

Basically, since the data owners may do not directly interact with the users, we delegate the owners' trust to a trusted authority (TA) and/or several local trusted authorities (LTAs) who are in charge of determining users' search privileges. We define a hierarchical relationship between the TA and LTAs (see Fig. 2). Define the "*local domain*" of an LTA to be the set of lower-level LTAs and users directly governed by it. The TA

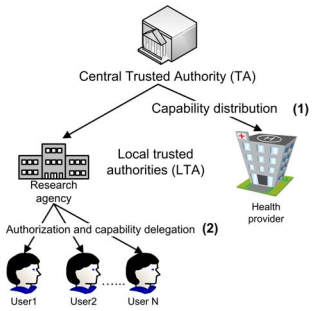


Fig. 2. Search authorization framework illustration.

runs Setup, and distributes some basic search capabilities via GenCap to each 2nd level LTA, *after this the TA can be offline most of the time*; while an ith level LTA runs DelegateCap to delegate the capabilities of itself to members in its local domain. A delegated capability must be more restrictive than its original one.

When a user requests a capability for query $\widehat{Q}$ from an LTA, the LTA checks whether a user either actually possesses the attribute value set $\mathbf{W}$ underlying the $\widehat{Q}$, or is "*eligible*" for those values. One way to achieve this is to maintain a database of attribute values for all users in the LTA's local domain. Alternatively, the LTA can issue to each user in its domain a set of credentials certifying the user's attribute values, and verifies those credentials upon a request for capability. In order to prove its authorization on a capability, a TA/LTA can issue an identity-based signature [31] on each capability it generated/delegated. The server has to verify that a received capability has a valid signature from a registered LTA before performing search for a user.

The rules of delegating search capabilities by the TA/LTAs to their local domains can be predefined by the system. The above captures the hierarchical relationship of access privileges of personnel in the real-world, and since the authorization tasks are distributed to each LTA, the system becomes more scalable.

Example. Let the TA be the public health agency of Boston, the 2nd level LTA be a hospital-A in Boston. TA gives the basic capability ($provider$ = "hospital-A") to hospital-A which indicates the basic restriction for searching on PHRs of patients who are treated by that hospital. The LTA can then delegate another capability like (age = "*") $\wedge$ ($illness$ = "diabetes") to a patient who is actually diagnosed to have diabetes by hospital-A, for patient matching. In contrast, a doctor can request to search for the specific type of disease she is treating on. Their capabilities should automatically inherit the restrictions of the LTA's.

IV. Basic Solution of APKS

A. Overview

The key challenge in building an APKS scheme is how to efficiently support multi-dimensional range queries and capability delegation at the same time. In this section, we exploit hierarchical predicate encryption (HPE) [30] to realize these goals. Since directly applying HPE does not result in acceptable efficiency due to the subset and range queries, we use HPE in a novel way that expresses a class of simple range queries using attribute hierarchy. The per-index search cost is $O(d \cdot m \cdot \log N)$ in the worst case, where N is the maximum size of $\mathcal{W}_i$, $d \ll N$ is the maximum allowed number of **OR** terms over all dimensions. The basic APKS scheme also supports efficient capability revocation.

B. Hierarchical Predicate Encryption

In HPE, given a ciphertext C for plaintext vector $\vec{x}$ and a secret key $\mathsf{sk}_{\vec{v}}$ for predicate vector $\vec{v}$, the decryption will succeed if $\vec{x} \cdot \vec{v} = 0$. There are two schemes of HPE in [30] —- for hierarchical and general delegation, respectively. We use the general delegation scheme in this paper, in which delegation is not limited to a specific hierarchical form defined in [30]. The algorithm definitions are given in the Appendix. In HPE, an $\ell+1$-th delegated secret key $\mathsf{sk}_{\vec{v}_\ell,\vec{v}_{\ell+1}}$ is more restrictive than its original one $\mathsf{sk}_{\vec{v}_\ell}$, in that $\mathsf{sk}_{\vec{v}_\ell,\vec{v}_{\ell+1}}$ can decrypt C only if $(\vec{v}_\ell \cdot \vec{x} = 0) \wedge (\vec{v}_{\ell+1} \cdot \vec{x} = 0)$.

Complexity of HPE: for n dimensional vectors, the lengths of ciphertext and secret key are both $n+3$ group elements, while decryption involves $n+3$ bilinear pairing operations.

C. Building APKS based on HPE

1) Basic Vector Representation for Indexes and Predicates

According to [21], one can represent the basic **AND**, **OR** predicates by first converting them into polynomial forms and then converting to vectors. In particular:

- For "$(Z_1 = w_1) \wedge (Z_2 = w_2)$": the polynomial is written as $p(Z_1, Z_2) = r(Z_1 - w_1) + (Z_2 - w_2)$, where $r \in_R \mathbb{F}_q$. Since $p(Z_1, Z_2) = rZ_1 + Z_2 - (rw_1 + w_2)$, the attribute vector is $\vec{x} = (Z_1, Z_2, 1)$ where we should substitute Z_i by the actual keyword z_i, while the predicate vector is $\vec{v} = (r, 1, -(rw_1 + w_2))$, and $p(Z_1, Z_2) = 0$ iff. $\vec{x} \cdot \vec{v} = 0$.
- For "$(Z_1 = w_1) \vee (Z_2 = w_2)$": $p(Z_1, x_2) = (Z_1 - w_1)(Z_2 - w_2) = Z_1Z_2 + w_2Z_1 + w_1Z_2 + w_1w_2$. Thus $\vec{x} = (Z_1Z_2, Z_1, Z_2, 1)$ and $\vec{v} = (1, w_2, w_1, w_1w_2)$.

Arbitrary CNF/DNF formulas can be represented by the combinations of the above, however, with exponential complexity [22]. The multi-dimensional keyword query considered in this paper is a special type of CNF formula, where conjunctions are across dimensions while disjunctions are within each dimension. This yields a tradeoff in expressiveness and complexity; however, it is versatile enough to support equality, subset and simple range queries, which are popular types of queries in reality.

In general, the polynomial corresponding to our choice of formulas can be expressed as a summation of m univariate polynomials:

$$p(Z_1, \cdots, Z_m) = r_1((Z_1 - w_{1,1}) \cdots (Z_1 - w_{1,d_1})) + \cdots + r_m((Z_m - w_{m,1}) \cdots (Z_m - w_{m,d_m})), \quad (1)$$

where $r_1, \cdots, r_m \in_R \mathbb{F}_q$. The plaintext vector is $\vec{x} = \psi(\vec{Z}) = (Z_1^{d_1}, \cdots, Z_1, Z_2^{d_2}, \cdots, Z_2, \cdots, Z_m^{d_m}, \cdots, Z_m, 1)$, where $\forall 1 \leq i \leq m, Z_i = z_i$, and the predicate vector is $\vec{v} = \phi(\widehat{Q}) = (c_{1,d_1}, \cdots, c_{1,1}, c_{2,d_2}, \cdots, c_{2,1}, \cdots, c_{m,d_m}, \cdots, c_{m,1}, c_0)$,

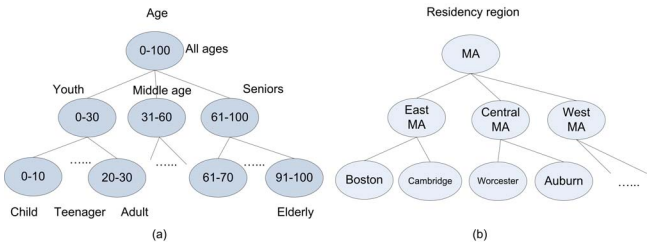


Fig. 3. Attribute hierarchies over different types of fields. (a): numerical attribute —- age. The path of leaf node "0-10" is ("0-100", "0-30", "0-10"), and "0-30" is a level-2 simple range. (b): non-numerical attributes —- residency region, which illustrates semantic containment.

where $c_{i,j}$ is the coefficient of Z_i^j and $c_0 = \sum_{i=1}^{m} c_{i,0}$. Apparently, the lengths of the vectors are both $n = \sum_{i=1}^{m} d_i + 1$. When each univariate polynomial's degree $d_i = d$, $n = md + 1$. For simplicity, we assume $\forall 1 \leq i \leq m, d_i = d$ in the following.

2) The Basic APKS Solution

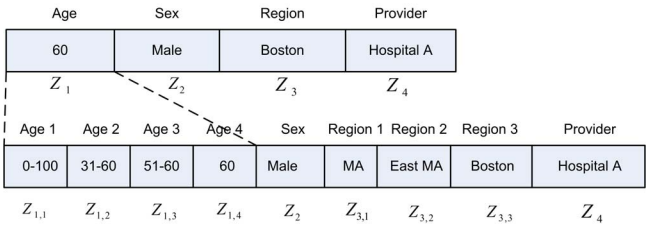
A straightforward method that directly converts a multi-dimensional query to its CNF form will yield an encryption/decryption complexity equal to $O(Nm)$, where N is the total number of possible attribute values in a field. The inefficiency is due to the large polynomial degree incurred by the many **OR** terms. To efficiently support these queries, it is beneficial to define attributes in a hierarchical way. The basic idea is similar to that of [33], however in order to support delegation we need to build the encrypted index in a specialized way. Note that, arbitrary range query is difficult to achieve with high efficiency using HPE, instead our solution supports a class of simple range queries.

Attribute Hierarchies.

Let Z be a numerical keyword field, an attribute hierarchy over Z is a balanced tree $T(Z)$, where each internal node represents a range that is the union of the ranges of its children nodes. For numerical field, we denote the range of a node id as an interval: $\mathsf{int}(id)$; for non-numerical ones, we define semantical range using semantical containment: if w_i is semantically contained by w, we say $w_i \sqsubset w$. For example, the attribute value "MA" semantically contains all the cities within the state MA and "flu" contains all kinds of illnesses that are flu.

In addition, for an attribute hierarchy $T(Z)$, we denote the node set in the l-th level as $T_l(Z)$, called a "*level-l attribute*". A node in $T_l(Z)$ is called a "*level-l keyword*", a.k.a. *level-l simple range*. For numerical attributes, multiple adjacent level-l simple ranges consist of a *level-l range*, while for non-numerical attributes each single node at level $l < k$ represents a *level-l semantic range*. The *path* from a leaf node z to the root is denoted as $\mathbb{P}(z)$, whereas for every node id in $\mathbb{P}(z)$, $z \in \mathsf{int}(id)$ or $z \sqsubset id$.

Fig. 3 shows two examples. Intuitively, we can use the hierarchies and simple ranges to make the range query more efficient. For example, the level-2 simple range $[0-30]$ represents all the integers from 0 to 30, so instead of converting query "$0 \leq$ *age* ≤ 30" to 30 equality terms **OR**ed together, there will be one equality term. In addition, we allow at most d **OR** combinations of multiple simple ranges of the same level like: ($region$ = "East MA" $\vee$ $region$ = "West MA").



(a) Index conversion. Age and region are hierarchical fields.

$$Q := (31 \leq Z_1 \leq 100) \wedge (Z_2 = \text{"Male"}) \wedge (Z_3 \in \text{"East MA"}) \wedge (Z_4 = \text{"HospitalA"})$$

$$Q'_{\mathbf{w}} := (Z_{1,2} = \text{"31-60"} \vee Z_{1,2} = \text{"61-100"}) \wedge (Z_2 = \text{"Male"}) \wedge (Z_{3,2} = \text{"EastMA"}) \wedge (Z_4 = \text{"HospitalA"})$$

(b) Query conversion. The rest of dimensions $Z_{1,1}, Z_{1,3}, Z_{1,4}, Z_{3,1}, Z_{3,3}$ are not shown since they are "don't care".

Fig. 4. Conversion of the index and query.

Note that, in this paper, we are not aiming at representing an arbitrary range $[s, t] \subseteq [1, N]$ for a numerical field. Although it can be represented by a collection of nodes from different levels [33], such combinations will incur high complexity. Thus we only consider the class of range queries containing simple ranges from one specific level over each dimension.

Nevertheless, this class of query is useful and flexible. On the one hand, the hierarchy on a numerical field can be well-designed to support semantic uses. For example, in Fig. 3 (a) the nodes are annotated with their semantic meanings for different age groups, such as "0-10" stands for childhood and "10-20" for teenager. On the other hand, a user can choose which ranges she wants to query, with different choices of granularity.

Generating the Encrypted Index and Capabilities. For a field with hierarchy defined, the basic idea is to include the path $\mathbb{P}(z)$ of its value z in the index, and the query over this dimension may consist of simple range query terms from one level. We define the following pre-processing steps.

- *Index conversion*. For each hierarchical field Z_i in an original index and whose maximum level is k, we expand it into k subfields (k is called *expansion factor*): $Z_{i,1}, \cdots, Z_{i,k}$, where the value set of $Z_{i,l}$ is $T_l(Z_i)$ and the value of $Z_{i,l}$ is the l-th element in $\mathbb{P}_l(z_i)$. Fig. 4 (a) shows that "age" is expanded into 4 subfields.
- *Query conversion*. An original query should be generated based on system-defined simple ranges. A user can select up to d l-level simple ranges or sematic ranges, either to compose a continuous range or discontinuous ranges. For example, if $d = 2$, an original query may look like $Q := (31 \leq Z_1 \leq 100)$. Then it is converted into a query over $Z_{i,l}$ and expressed in CNF form: $\widehat{Q} := (Z_{1,2} =$ "31-60" $\vee\, Z_{1,2} =$ "61-100"). This is shown in Fig. 4 (b). The number of dimensions of a query becomes m'.

The converted plaintext and query are fed into $\mathsf{GenIndex}$ and GenCap algorithms to generate vectors $\vec{x}$ and $\vec{v}$, respectively. The formal definitions of the algorithms are in Fig. 5.

Example. The TA generates a capability $T_{f_{\vec{v}_1}}$ for $Q_1 :=$

- Setup(1^κ). Given $n = \sum_{i=1}^{m'} d_i$, ϕ, ψ as in Sec. IV-C.1, run HPE-Setup(1^κ), output $PK = (\mathsf{pk}, \phi, \psi)$, and $MSK = \mathsf{msk}$.
- GenIndex(PK, $\vec{Z}$). First convert the index according to Sec. IV-C.3. Set $\mathsf{m} = \mathsf{Msg}||0^\lambda$, where $\mathsf{Msg} \in_R \{0,1\}^{\kappa-\lambda}$, and compute $\vec{x} = \psi(\vec{Z})$. Output $E(\vec{Z})$ =HPE-Enc(PK, $\vec{x}$, m).
- GenCap(PK, MSK, Q). First convert Q to its CNF form: $\widehat{Q}$ using the method above, and compute $\vec{v} = \phi(\widehat{Q})$. Then run HPE-GenKey($PK$, MSK, $\vec{v}$), and output $T_Q = \mathsf{sk}_{\vec{v}}$.
- Search(PK, T_Q, $E(\vec{Z})$). Run HPE-Dec(PK, $\mathsf{sk}_{\vec{v}} = T_Q$, $C = E(\vec{Z})$), and returns true if HPE-Dec$(PK, \mathsf{sk}_{\vec{v}}, C) = \mathsf{Msg}||0^\lambda$ for some Msg; Otherwise return false.
- DelegateCap(PK, T_{Q_1}, Q_2). Convert Q_2 to $\widehat{Q}_2$, and set $\mathsf{sk}_{\vec{v}_1,\cdots,\vec{v}_\ell} = T_{Q_1}$ and $\vec{v}_{\ell+1} = \phi(\widehat{Q}_2)$, run HPE-Delegate($\mathsf{sk}_{\vec{v}_1,\cdots,\vec{v}_\ell}$, $\vec{v}_{\ell+1}$), outputs $T_{\vec{v}_1,\cdots,\vec{v}_{\ell+1}} = \mathsf{sk}_{\vec{v}_1,\cdots,\vec{v}_{\ell+1}}$. Note that $T_{\vec{v}_1,\cdots,\vec{v}_{\ell+1}}$ corresponds to query $Q_1 \wedge Q_2$.

Fig. 5. The details of basic APKS solution.

(*provider* = "Hospital A") $\wedge$ (*region* = "East MA") and gives it to hospital A's LTA. When a physician Peter in Hospital A requests for a capability for $Q_2 := ((age > 60) \wedge (illness =$ "Diabetes")), Hospital A converts Q_2 to $\vec{v}_2$ and generates a delegated capability $T_{f_{\vec{v}_1},\vec{v}_2}$ based on $T_{f_{\vec{v}_1}}$ and $\vec{v}_2$, which actually gives Peter the capability for restricted query (Q_1 **AND** Q_2). Hospital A can generate delegated capabilities for any of the "don't care" dimensions in Q_1; it can also delegate a subset of her capabilities on existing dimensions, such as (*region* = "Boston") $\wedge \cdots$ to a user.

Revocation. Our solution efficiently supports revocation by adding another time attribute in the indexes and capabilities, in which the former indicates the create time of an owner's index, the latter specifies a user's authorized search period. The periods can be efficiently expressed by simple ranges. For example, the hierarchy could be "year-month-week-day", and a capability will look like "(*time* = [Jan.2010 − Jun.2010]) $\wedge$ $\cdots$" which can search the PHRs created during Jan. 2010 and Jun. 2010. When an owner updates her PHR and the index, she may also change the time value. A capability with expired time period cannot be used to search newer indexes, to do that a user must obtain a new capability from an LTA.

Complexity. For our solution, there are at most $k = \log_2 N$ sub-fields for each hierarchical field, thus when every field is hierarchical, $m' = mk$ and the length of plaintext and predicate vectors is $O(m' \cdot d) = O(m \cdot d \cdot \log N)$.

V. Enhanced Solution: Towards Query Privacy

The basic solution achieves index privacy due to the security of the HPE scheme. However, it cannot not prevent the server from knowing the underlying query within a capability by launching the *dictionary attack*. This is because the HPE is a public key based predicate encryption scheme; given a capability T_Q for some query Q and an attribute universe $\mathcal{W}$, the server can try to encrypt all possible indexes $\vec{Z}$ by brute-force through all combinations of keywords in each field. It tests each of those encrypted indexes against T_Q; if T_Q matches with a ciphertext $E(\vec{Z})$, the server can deduce Q from $Q(\vec{Z}) = 1$. For example, if $Q := (Z_1 = z_1 \wedge Z_2 = z_2)$ the complexity of such attack is only $|\mathcal{W}_1| \times |\mathcal{W}_2|$. From Q, it can also learn about the plaintext index corresponding to the returned records. The above attack compromises query privacy, or predicate privacy [32]. It is claimed that it is inherently impossible to achieve predicate privacy under the pure public-key setting [32]. In this section, we propose APKS$^+$, which prevents the dictionary attack with the help of auxiliary infrastructure.

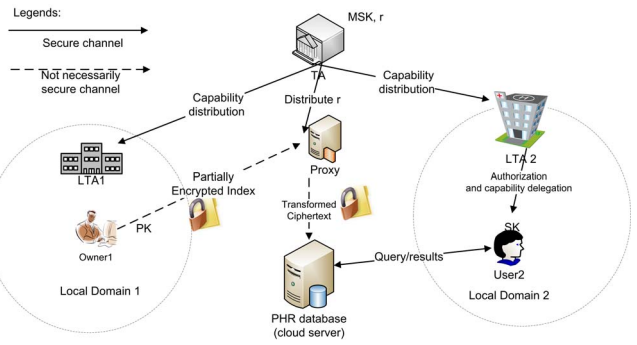


Fig. 6. The enhanced framework for APKS$^+$ that preserves query privacy: single proxy case.

A. Solution Overview

To thwart the dictionary attack, one need to prevent the adversary from generating valid ciphertexts based on PK and a set of meaningful keywords. Our main idea is to involve an additional secret r during encryption and decryption which is hidden from the adversary. Since distributing r to all the users will increase the risk of key exposure, r is kept by auxiliary proxy server(s) who would transform the partial encryption by PK on the encryptors' behalves. It does not require interactions between the owners/encryptors and the users.

Our idea is partly inspired by Zhu et al [40] who addressed predicate privacy in PEKS. However, our solution is different in two key aspects: (1) The application scenario and system framework in this paper are different. PEKS is suitable for email filtering applications and in PEKS each user can generate unrestricted search capabilities by herself. (2) We aim at providing query privacy for HPE and our construction is different from theirs.

B. Main Design of APKS$^+$

1) Enhanced System Framework

For the threat model, we assume that the proxy server(s) are semi-trusted, and an adversary cannot control both cloud server and proxy server(s) at the same time. We argue this is reasonable in practice since a proxy server can be well protected by the organization that owns it (e.g., the TA's organization). Also, we assume the cloud server do not launch active attacks such as probe-response attack [8] which needs a large amount of partial ciphertexts be sent to the proxy, as there exist some detection mechanism (e.g., traffic monitoring).

We depict the enhanced system framework in Fig. 6. The TA generates a random secret r in addition to MSK and PK, the proxy is given r, which is also embedded into the capabilities for the LTAs and users. An owner partially encrypts a PHR

- $\mathsf{HPE}^+ - \mathsf{Setup}(1^\kappa)$. Generate param of a $n+3$ dimensional vector space, and X, $\mathbb{B}$, $\widehat{\mathbb{B}}$, $\mathbb{B}^*$ in the same way as in $\mathsf{HPE\text{-}Setup}$. Pick $r \in_R \mathbb{F}_q - \{0\}$, set basis $\widetilde{\mathbb{B}}^* := r\mathbb{B}^* = \{r\boldsymbol{b}_1^*, \cdots, r\boldsymbol{b}_{n+3}^*\}$, and output $\mathsf{pk} := (1^\kappa, \mathsf{param}, \widehat{\mathbb{B}})$ (same), $\mathsf{msk} := (X, \widetilde{\mathbb{B}}^*)$ (*changed from* $(X, \mathbb{B}^*)$). The TA distributes r^{-1} to the proxy server via a secure channel.
- $\mathsf{HPE}^+ - \mathsf{PartialEnc}(\mathsf{pk}, \vec{x} := (x_1, \cdots, x_n), \mathsf{m})$: Executed by the encryptor, which is the same with $\mathsf{HPE\text{-}Enc}$ and return $C := (\boldsymbol{c}_1, c_2)$.
- $\mathsf{HPE}^+ - \mathsf{ProxyEnc}(r^{-1}, C)$: Executed by a proxy. Parse C as $(\boldsymbol{c}_1, c_2)$, compute $\boldsymbol{c}_1' = r^{-1}\boldsymbol{c}_1$. Output $C' = (\boldsymbol{c}_1', c_2)$.
- $\mathsf{HPE}^+ - \mathsf{GenKey}(\mathsf{pk}, \mathsf{msk}, \vec{v}_1 := (v_{1,1}, \cdots, v_{1,n}))$: Basically the same as $\mathsf{HPE\text{-}GenKey}$, difference is that now $\mathsf{sk}_{\vec{v}_1}$ is generated using basis $\widetilde{\mathbb{B}}^*$ instead of $\mathbb{B}^*$. A user delivers the key (trapdoor) to the cloud server via a secure channel.
- $\mathsf{HPE}^+ - \mathsf{Dec}(\mathsf{pk}, \boldsymbol{k}_{\mathsf{dec}}^*, \boldsymbol{c}_1', c_2)$: The same as $\mathsf{HPE\text{-}Dec}$.
- $\mathsf{HPE}^+ - \mathsf{Delegate}(\mathsf{pk}, \vec{\boldsymbol{k}}_\ell^*, \vec{v}_{\ell+1} := (v_{\ell+1,1}, \cdots, v_{\ell+1,n}))$: The same as $\mathsf{HPE\text{-}Delegate}$.

Fig. 7. The details of HPE^+ scheme.

index using PK, which will be transformed by the proxy before storing it on the cloud server. To learn r an adversary must compromise both the cloud server and the proxy.

To increase the attack-resiliency, a second choice is to employ multiple proxies (not shown) where each of them can be hosted by a local domain. And the secret r will be divided into $r_1, \cdots, r_P$ s.t. $r = r_1 r_2 \cdots r_P$ where each r_i is distributed to one proxy. In this case, a partial ciphertext needs to traverse all P proxies before sent to the cloud server, with the cost of increased delay. Our solution supports both design choices.

2) Scheme Details

In Fig. 7, we present an enhanced HPE^+ scheme under the first case (one proxy). The application of it to APKS^+ is the same as in the basic solution. We will denote a vector of points on elliptic curve as $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{k}$; $r\boldsymbol{b}$ stands for point multiplication of r with each element in $\boldsymbol{b}$, and $\mathbb{B}$ for basis in a vector space.

Correctness. Since r^{-1} is integrated into the transformed ciphertext while r is embedded into the user secret key, during decryption they cancel out each other and the result will be correct. Please refer to our technical report [25] for details.

VI. Security Analysis

A. Security of the Basic Solution

Index Privacy. The original HPE scheme has been proven selectively attribute-hiding secure under the standard model [30]. That is, two attribute/plaintext vectors encrypted in two ciphertexts are indistinguishable for a computationally bounded adversary (e.g., cloud server), as long as the adversary does not obtain a trapdoor that distinguishes the two ciphertexts. In other words, the adversary learns nothing about the index $\vec{Z}$ underlying a ciphertext in polynomial time as long as the user-submitted capability T_Q satisfies $Q(\vec{Z}) = 0$. On the other hand, when $Q(\vec{Z}) = 1$, T_Q will reveal information about the underlying query, and combined with search results the adversary can guess the returned indexes with non-negligible probability. Thus the basic solution provides a weak form of index privacy, while does not achieve query privacy.

B. Security of the Enhanced Solution

Query Privacy. We claim that the APKS^+ achieves both index privacy and query privacy under our threat model. The TA publishes $\mathsf{pk} := (1^\kappa, \mathsf{param}, \widehat{\mathbb{B}})$ where $\widehat{\mathbb{B}}$ is based on $\mathbb{B}$, while the ciphertexts (after proxy transformation) and capabilities are actually based on $\widetilde{\mathbb{B}}$ and $\widetilde{\mathbb{B}}^*$, respectively, where $\widetilde{\mathbb{B}} = r^{-1}\mathbb{B}$ and $\widetilde{\mathbb{B}}^* = r\mathbb{B}^*$. On the one hand, r is kept by the TA and r^{-1} are kept by the proxy server(s). On the other hand, even if the adversary eavesdrops the communication between owners and proxies and obtains pairs of partial ciphertexts $\boldsymbol{c}_1$ and their corresponding transformed ciphertexts $\boldsymbol{c}_1' = r^{-1}\boldsymbol{c}_1$, because of the intractability of Elliptic Curve Discrete Logarithm Problem (ECDLP), the adversary cannot get r. Thus, without the ability to generate a legitimate ciphertext for meaningful index, the adversary cannot launch the dictionary attack.

In addition, since we assume the cloud server cannot launch probe-response attack, it cannot deliver a large number of partially encrypted, guessed indexes to the proxy and receive the transformed ciphertexts. Moreover, if the cloud server colludes with a malicious user, and if the user launches the probe-response attack, it has a high chance to be detected when both the attribute universe in each dimension and the number of dimensions in the victim user's query are large.

Statistical Attacks. When the adversary has some background information such as the probability distribution of appearance of each keyword in all the indexes, it will be able to guess the underlying query keyword of a search capability. A natural countermeasure is to require each query must contain no less than a certain number of dimensions, please refer to [25] for more details on this.

VII. Performance Evaluation

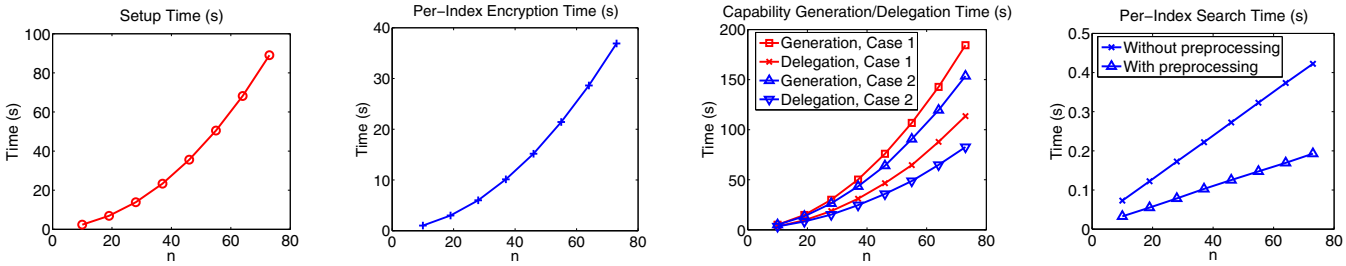
We have implemented HPE and the basic solution of APKS, using the Pairing-Based Cryptography (PBC) Library [29]. We run the experiments on a server running Linux with a 32-bit, 3.4GHz Pentium D CPU. We adopt the type A elliptic curve parameter [29], where the group order q is 160-bits, which provides 80-bit security strength equivalently.

A. Experimental Setup

We carry out a proof-of-concept performance demonstration of our solution using the Nursery data set from the UCI Machine Learning Repository [17], which has also been used in previous works on searchable encryption [37]. The data set features categorical attributes and has 8 attributes where each attribute has up to 5 values. Each attribute is treated as a dimension, and each attribute value is converted into elements in $\mathbb{F}_q$ using SHA1 hash algorithm. The data set contains 12,960 instances (rows) and 9 dimensions (columns).

B. Results

In the following we evaluate the computation, communication and storage overhead of APKS. As far as we know, there are few existing works on multi-dimensional searchable encryption with experimental results. We will compare the efficiency of APKS with the $MRQED^D$ range query scheme in [33], where the running times were estimated based on



(a) Setup time. (b) Per-index encryption time. (c) Capability generation/delegation. (d) Per-index search processing time.

Fig. 8. Time efficiency of APKS.

benchmark results. As a result, although APKS is slower than $MRQED^D$ in setup, encryption and capability generation, it is much faster in search operations.

1) Setup

The major overhead of the setup process is generating the bases $\widehat{\mathbb{B}}$ and $\mathbb{B}^*$, which involves $O(n_0^2) = O(n^2)$ exponentiations (point multiplications) each, where $n_0 = n + 3 = \sum_{i=1}^{m'} d_i + 4$ is the dimension of the ECC vector spaces in HPE and n is lengths of $\vec{x}, \vec{v}$. In Fig. 8 (a), we plot the average time for setup against n. The setup time is about 40s when $n = 46$, which is an one-time cost. For storage overhead, although the size of the base field for $\mathbb{G}_1, \mathbb{G}_2$ is 512-bits, an elements in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ can be represented by 65B in compressed form, while q is 20B. Thus the total size of PK is $65[n_0(n_0-1)+3]$B, that of MSK is $n_0^2(65+20) = 85n_0^2$B. When $n = 46$, these equal to 153KB and 204KB, respectively. For the $MRQED^D$ scheme, setup takes $O(n)$ exponentiations. When $n = 46$, the setup time and key sizes equal to 4.6s, 22.5KB and 22.5KB, respectively.

2) Encrypted Index Generation

Next we compute the average encryption time for each index (row). We assume $d_i = d, \forall 1 \le i \le m'$, either fix $m' = 9$ and vary d from 1 to 5, or fix $d = 1$ and vary m' from 9 to 72, and confirmed the time depends only on $m'd$, i.e, n. Note that, for the latter case, each original index field is duplicate by factors of $2, 3, \cdots, 8$ to mimic the expansions of hierarchical attributes in APKS. The results are presented in Fig. 8 (b); again, the per-index encryption time scales as $O(n_0^2)$. When $n = 46$, encrypting an index takes about 15s. Since we are aiming at searching on encrypted PHR in the public domain and each owner generates one (or a few) index for her PHR documents, this overhead is acceptable in practice. The size of an encrypted index is $65(n_0 + 1)$, which equals to merely 3.25KB when $n = 46$. For the $MRQED^D$ scheme, encryption takes $O(n)$ exponentiations. When $n = 46$, the time and ciphertext size equal to 2.3s and 11.6KB, respectively.

3) Capability Generation and Delegation

We carry out two sets of experiments here, and show timing results of both capability generation and the first level delegation in Fig. 8 (c). In the first set, we assume there are no hierarchical attributes ($k = 1$) and the input query covers all m' dimensions. Also, we randomly select d keywords from the keyword universe in each dimension to form a query. In this case, the predicate vector $\vec{v}$ does not contain element $0 \in \mathbb{F}_q$, which can be regarded as the worst case. It can be seen that, although the direct capability generation takes relatively long time, the delegation takes less time. The former is performed by the TA; since it is usually an one-time operation, it is not a burden. The latter delay is experienced by a Level-2 LTA and users under a Level-1 LTA, which is faster. Note that, the capability generation/delegation times both scale as $O(n_0^2)$.

In the second set, the query over some of the dimensions are set to "don't care", so as to represent the more realistic case. Each original field in the index is expanded into k subfields ($m' = 9k$), a query involves at most 9 non-don't care fields instead of m'. We set $d_i = d = 1$ and change the expansion factor k from 1 to 8. In Fig. 8 (c), one can observe that both the capability generation and delegation times increase slower with n than in the first set, which is due to the "don't care" terms. When $n = 46$, delegating a capability takes about 35s. Finally, the total size of a capability is $65[n_0^2 + (l+3)n_0]$B, where l is the number of times a capability has been delegated. When $n = 46, l = 1$, this equals to 169KB. For the $MRQED^D$ scheme, capability generation takes $O(n)$ exponentiations. When $n = 46$, the time and capability sizes are 2.3s and 14.4KB, respectively.

4) Search

We show in Fig. 8 (d) the average search processing time on single encrypted index under different n values. It can be seen that the search is much faster than encryption and is linear to n, since it only takes $n + 3$ pairing operations. In our benchmark tests, it takes 5.5ms for a single pairing operation under type-A parameter without preprocessing, and 2.5ms with preprocessing. Based on the average search time, we estimate the total search time for the Nursery data set, by multiplying the former with 12,960. The results are given in Table III. When $n = 46$, it takes about 27min to search; although it seems large, we argue that this is still acceptable for practical use, especially for delay-tolerant applications like profile matching, medical research etc. Also, if the cloud server have multiple processors the search computation can be done in a paralleled way. For the $MRQED^D$ scheme, per-index search takes $O(m \log N) = 5n$ pairings. When $n = 46$, the time equals to 0.59s with preprocessing, 5 times of ours.

C. Discussion

In the above results, for the same n value, there are multiple design choices of m, k and d. This actually provides a tradeoff

n	10	19	28	37	46	55	64	73
Time (s)	424	714	1016	1330	1625	1911	2194	2498

TABLE III
PROJECTED TOTAL SEARCH TIMES FOR THE NURSERY DATA SET (12,960 INDEXES) UNDER DIFFERENT n VALUES, WITH PAIRING PREPROCESSING.

between the number of index fields and the maximum number of **OR** terms. In general, the larger the expansion factor k, the more expressive is a simple range query over each dimension, and the maximum number of OR terms d can be made smaller. For example, for the Nursery data set, if $n = 46$, we can have $m = 9, d = 5$ and $k = 1$, or $m = 9, d = 1$ while $k = 5$.

Moreover, in practice not all the attributes are hierarchical. For example, "age", "region","illness" are but "provider", "sex" etc are not. If Z_1, Z_2, Z_3 out of $Z_1 \cdots, Z_9$ are hierarchical, and $k = 9$, the maximum number of **OR** terms over each sub-field is 1, but 3 for $Z_4, \cdots, Z_9$, we have $n = 3 \times 9 + 6 \times 3 + 1 = 46$. Thus, the case of $n = 46$ is a representative one.

VIII. RELATED WORK

A. Searchable Encryption with Authority Generated Capabilities

In applications such as outsourced private databases, usually the (sole) owner of the outsourced data plays the role of the authority who generates search capabilities for users. Song et al [35] proposed the first practical symmetric key cryptography (SKC) based searchable encryption scheme. Later, various schemes [18], [14], [15], [37] were proposed where a searchable encrypted index is usually created, so that the encrypted documents can be searched given a capability. The first public key cryptography (PKC)-based searchable encryption scheme was proposed by Boneh et al [10]. Recently Wang et al [36] investigated secure ranked search over encrypted cloud data. The above schemes are efficient in general, but are limited to single-keyword queries which are inadequate for real-world PHR search applications.

To support more complex queries, conjunctive keyword search schemes [19], [5] over encrypted data have been proposed. Recently a more general approach, predicate encryption [21], [32] was proposed that supports inner-products. They can potentially support arbitrary query types including CNF/DNF formulas, however, with exponential complexity. In this paper, we consider the multi-dimensional keyword search supporting a subset of CNF formulas with equality, subset and a class of simple range queries. Our work is more related to [11], [33], where multi-dimensional arbitrary range queries are considered. In hidden vector encryption [11], the complexity of conjunctive subset and range query is $O(Nm)$, which is much less efficient than our APKS. Moreover, they do not support capability delegation.

Recently predicate encryption with delegation capabilities was proposed in [30], [22] and [34]. However, these schemes are under public key setting and do not achieve query (predicate) privacy [32] per se. Camenisch et al [12] proposed a PKC-based searchable encryption scheme with an authorization process that is oblivious to the TA, which is orthogonal to the problems addressed in our work.

B. Searchable Encryption with User Generated Capabilities

In this category, each user can generate any search capabilities of her choice. This includes single-user scheme such as PEKS [10], and "multi-user" schemes such as Curtmola et al [15], Bao et al [6], and Dong et al [16].

The main advantage of schemes in this category is they obviate the overhead for users to acquire search capabilities. However, search authorization is intrinsically difficult to achieve since it is contradictory with user-generated capabilities. An exception is Hwang and Lee's work [20], who proposed a public key encryption scheme with conjunctive search (PECK) with main applications to group email filtering. Unfortunately, the ciphertext length grows linearly with the number of authorized users. In contrast, in this paper search authorization is based on users' attributes and enforced by LTAs indirectly, while the length of an encrypted index is independent of the list of authorized users.

C. Predicate Privacy in Searchable Encryption

Shen et al [32] proposed a SKC-based predicate encryption scheme with predicate privacy. However, in a SKC-based scheme with an encryption key one can generate any search capabilities, which is undesirable for search authorization. Under public key setting, [4] proposed a key refreshing solution to randomize the original keywords used in generating the trapdoors in PEKS, but that requires a user to share a secret with all potential encryptors. In contrast, in our enhanced solution some secrets are kept by proxies, thus it does not incur any interaction between owners and users.

IX. CONCLUDING REMARKS

In this paper, we address the problem of authorized private keyword searches over encrypted data in cloud computing, where multiple data owners encrypt their records along with a keyword index to allow searches by multiple users. To limit the exposure of sensitive information due to unrestricted query capabilities, we propose a scalable, fine-grained authorization framework where users obtain their search capabilities from local trusted authorities according to their attributes. We then propose two novel solutions for APKS over encrypted data based on HPE, where in the first one we enhance the search efficiency using attribute hierarchy, and in the second we enhance the query privacy via the help of proxy servers. Our solutions also support efficient multi-dimensional range query, search capability delegation and revocation. In addition, we implement our solution on a modern workstation; the results show that APKS achieves reasonable search performance.

Acknowledgements. This work was supported in part by the US National Science Foundation under grants CNS-0716306, CNS-0831628, and CNS-0746977. We would like to thank Prof. Joshua. D. Guttman for his helpful discussions.

REFERENCES

[1] Googlehealth. https://www.google.com/health/.
[2] At risk of exposure – in the push for electronic medical records, concern is growing about how well privacy can be safeguarded, 2006.
[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing, Feb 2009.

[4] J. Baek, R. Safavi-Naini, and W. Susilo. Public key encryption with keyword search revisited. In *Proceedings of ICCSA, Part I*, ICCSA '08, pages 1249–1259, 2008.

[5] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Proc. of ICICS'05*, 2005.

[6] F. Bao, R. H. Deng, X. Ding, and Y. Yang. Private query on encrypted data in multi-user settings. In *ISPEC'08*, pages 71–85, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *CCSW '09*, pages 103–114, 2009.

[8] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In *Proceedings of the 14th conference on USENIX Security Symposium*, pages 13–13, 2005.

[9] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE S& P '07*, pages 321–334, 2007.

[10] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. of EUROCRYP'04*, 2004.

[11] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Proc. of TCC'07*, pages 535–554, 2007.

[12] J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy. Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In *PKC'09*, pages 196–214. Springer-Verlag.

[13] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *IEEE INFOCOM*, 2011.

[14] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proc. of ACNS'05*, pages 442–455, 2005.

[15] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proc. of ACM CCS'06*, 2006.

[16] C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In *Journal of Computer Security*, 2010.

[17] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[18] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/.

[19] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In *ACNS 04*, pages 31–45. Springer-Verlag, 2004.

[20] Y. Hwang and P. Lee. Public key encryption with conjunctive keyword search and its extension to a multi-user system. In *Pairing-Based Cryptography (Pairing 2007)*, volume 4575 of *LNCS*, pages 2–22. 2007.

[21] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT'08*, pages 146–162, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT 2010*.

[23] M. Li, N. Cao, S. Yu, and W. Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *IEEE INFOCOM*, 2011.

[24] M. Li, W. Lou, and K. Ren. Data security and privacy in wireless body area networks. *IEEE Wireless Communications Magazine*, Feb. 2010.

[25] M. Li, S. Yu, N. Cao, and W. Lou. Authorized private keyword search over encrypted data in cloud computing. *Technical report, http://ece.wpi.edu/ mingli/*, Mar. 2011.

[26] 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 *SecureComm'10*, pages 89–106, Sept. 2010.

[27] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management: extended abstract. In *ACM CCS*.

[28] R. Lu, X. Lin, X. Liang, and X. S. Shen. Secure handshake with symptoms-matching: The essential to the success of mhealthcare social network. *Mobile Netw. Appl.*, To appear.

[29] B. Lynn. The pbc library. http://crypto.stanford.edu/pbc/.

[30] T. Okamoto and K. Takashima. Hierarchical predicate encryption for inner-products. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 214–231. 2009.

[31] K. Paterson and J. Schuldt. Efficient identity-based signatures secure in the standard model. In L. Batten and R. Safavi-Naini, editors, *Information Security and Privacy*, LNCS. 2006.

[32] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *Theory of Cryptography*, volume 5444 of *LNCS*, pages 457–473. 2009.

[33] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, SP '07, pages 350–364, 2007.

[34] E. Shi and B. Waters. Delegating capabilities in predicate encryption systems. In *ICALP '08*, pages 560–578, 2008.

[35] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proc. of IEEE S & P '00*, 2000.

[36] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *Proc. of ICDCS'10*, 2010.

[37] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In *Proc. of ESORICS'06*, pages 479–495, 2006.

[38] S. Yu, K. Ren, W. Lou, and J. Li. Defending against Key Abuse Attacks in KP-ABE Enabled Broadcast Systems. In *SecureComm'09*.

[39] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *IEEE INFOCOM'10*, 2010.

[40] B. Zhu, B. Zhu, and K. Ren. Peksrand: Providing predicate privacy in public-key encryption with keyword search. Cryptology ePrint Archive, Report 2010/466, 2010.

APPENDIX A
A BRIEF REVIEW OF HPE SCHEME [30]

Here we review the HPE scheme for general delegation. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear map, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of prime order q with g_1, g_2 be generators, respectively, and $g_T := e(g_1, g_2)$. Define n-dimensional vector spaces $\mathbb{V} := \mathbb{G}_1 \times \cdots \times \mathbb{G}_1$, and $\mathbb{V}^* := \mathbb{G}_2 \times \cdots \times G_2$ whose elements are vectors $\boldsymbol{x} := (g_1^{x_1}, \cdots, g_1^{x_n})$ and $\boldsymbol{y} := (g_2^{y_1}, \cdots, g_2^{y_n})$, respectively, where $x_i, y_i \in \mathbb{F}_q$. Pairing operation between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is $e(\boldsymbol{x}, \boldsymbol{y}) := \prod_{i=1}^n e(g_1^{x_i}, g_2^{y_i}) = e(g_1, g_2)^{\sum_{i=1}^n x_i y_i} = g_T^{\vec{x} \cdot \vec{y}}$.

Define canonical bases $\mathbb{A} := (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n)$ of $\mathbb{V}$, where $\boldsymbol{a}_i := (1, \cdots, 1, g_1, 1, \cdots, 1)$ (all identity elements except the i-th position), and $\mathbb{A}^* := (\boldsymbol{a}_1^*, \cdots, \boldsymbol{a}_n^*)$ of $\mathbb{V}^*$ is defined similarly. Canonical basis $\mathbb{A}$ can be changed to basis $\mathbb{B} := (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ by $\mathbb{B} = X \cdot \mathbb{A}$ where $X := (\chi_{i,j}) \in_R GL(n, \mathbb{F}_q)$, and $\mathbb{B}^* = (X^T)^{-1} \cdot \mathbb{A}^*$. $\mathbb{B}$ and $\mathbb{B}^*$ are dual orthonormal bases.

HPE-Setup(1^κ): $\mathsf{param} := (q, \mathbb{V}, \mathbb{V}^*, \mathbb{G}_T, \mathbb{A}, \mathbb{A}^*)$ where $\mathbb{V}, \mathbb{V}^*$ are $n+3$ dimensional spaces. Generate X, $\mathbb{B}$, $\mathbb{B}^*$ as above; $\boldsymbol{d}_{n+1} := \boldsymbol{b}_{n+1} + \boldsymbol{b}_{n+2}$, $\widehat{\mathbb{B}} := (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n, \boldsymbol{d}_{n+1}, \boldsymbol{b}_{n+3})$, return $\mathsf{pk} := (1^\kappa, \mathsf{param}, \widehat{\mathbb{B}})$, $\mathsf{msk} := (X, \mathbb{B}^*)$.

HPE-GenKey(pk, msk, $\vec{v}_1 := (v_{1,1}, \cdots, v_{1,n})$): Pick $\sigma_{\mathsf{dec}}, \eta_{\mathsf{dec}}, \sigma_{\mathsf{ran},1}, \sigma_{\mathsf{ran},2}, \eta_{\mathsf{ran},1}, \eta_{\mathsf{ran},2}, \sigma_{\mathsf{del},j}, \eta_{\mathsf{del},j}, \varphi \in_R \mathbb{F}_q$, for $j = 1, \cdots, n$. Output $\mathsf{sk}_{\vec{v}_1} := \vec{\boldsymbol{k}}_1^* := (\boldsymbol{k}_{1,\mathsf{dec}}^*, \boldsymbol{k}_{1,\mathsf{ran},1}^*, \boldsymbol{k}_{1,\mathsf{ran},2}^*, \boldsymbol{k}_{1,\mathsf{del},1}^*, \cdots, \boldsymbol{k}_{1,\mathsf{del},n}^*)$, where:
$\boldsymbol{k}_{1,\mathsf{dec}}^* := \sigma_{\mathsf{dec}}(\sum_{i=1}^n v_{1,i}\boldsymbol{b}_i^*) + \eta_{\mathsf{dec}}\boldsymbol{b}_{n+1}^* + (1 - \eta_{\mathsf{dec}})\boldsymbol{b}_{n+2}^*$;
$\boldsymbol{k}_{1,\mathsf{ran},j}^* := \sigma_{\mathsf{ran},j}(\sum_{i=1}^n v_{1,i}\boldsymbol{b}_i^*) + \eta_{\mathsf{ran},j}\boldsymbol{b}_{n+1}^* - \eta_{\mathsf{ran},j}\boldsymbol{b}_{n+2}^*$ $(j = 1, 2)$;
$\boldsymbol{k}_{1,\mathsf{del},j}^* := \sigma_{\mathsf{del},j}(\sum_{i=1}^n v_{1,i}\boldsymbol{b}_i^*) + \varphi\boldsymbol{b}_j^* + \eta_{\mathsf{del},j}\boldsymbol{b}_{n+1}^* - \eta_{\mathsf{del},j}\boldsymbol{b}_{n+2}^*$ $(j = 1, \cdots, n)$.

HPE-Enc(pk,$\vec{x} := (x_1, \cdots, x_n)$, $\mathsf{m} \in \mathbb{G}_T$): Pick $\delta_1, \delta_2, \zeta \in_R \mathbb{F}_q$, return $C := (\boldsymbol{c}_1, c_2)$ where $\boldsymbol{c}_1 := \delta_1(\sum_{i=1}^n x_i\boldsymbol{b}_i) + \zeta\boldsymbol{d}_{n+1} + \delta_2\boldsymbol{b}_{n+3}$, $c_2 := g_T^\zeta \mathsf{m}$.

HPE-Dec(pk, $\boldsymbol{k}_{\mathsf{dec}}^*$, $\boldsymbol{c}_1, c_2$): $\mathsf{m}' := c_2 / e(\boldsymbol{c}_1, \boldsymbol{k}_{\mathsf{dec}}^*)$, return m'.

HPE-Delegate(pk,$\vec{\boldsymbol{k}}_\ell^*$, $\vec{v}_{\ell+1} := (v_{\ell+1,1}, \cdots, v_{\ell+1,n})$): Output $\mathsf{sk}_{\vec{v}_1,\cdots,\vec{v}_{\ell+1}} := \vec{\boldsymbol{k}}_{\ell+1}^* := (\boldsymbol{k}_{\ell+1,\mathsf{dec}}^*, \boldsymbol{k}_{\ell+1,\mathsf{ran},1}^*, \cdots, \boldsymbol{k}_{\ell+1,\mathsf{ran},\ell+2}^*, \boldsymbol{k}_{\ell+1,\mathsf{del},1}^*, \cdots, \boldsymbol{k}_{\ell+1,\mathsf{del},n}^*)$, where:
$\boldsymbol{k}_{\ell+1,\mathsf{dec}}^* := \boldsymbol{k}_{\ell,\mathsf{dec}}^* + \sum_{i=1}^\ell \alpha_{\mathsf{dec},i}\boldsymbol{k}_{\ell,\mathsf{ran},i}^* + \sigma_{\ell+1,\mathsf{dec}}(\sum_{i=1}^n v_{\ell+1,i}\boldsymbol{k}_{\ell,\mathsf{del},i}^*)$, $(\alpha_{\mathsf{dec},i}, \sigma_{\ell+1,\mathsf{dec}} \in_R \mathbb{F}_q, i = 1, \cdots, \ell)$;
$\boldsymbol{k}_{\ell+1,\mathsf{ran},j}^* := \sum_{i=1}^\ell \alpha_{\mathsf{ran},i}\boldsymbol{k}_{\ell,\mathsf{ran},i}^* + \sigma_{\ell+1,\mathsf{ran},j}(\sum_{i=1}^n v_{\ell+1,i}\boldsymbol{k}_{\ell,\mathsf{del},i}^*)$, $(\alpha_{\mathsf{ran},i}, \sigma_{\ell+1,\mathsf{ran},j} \in_R \mathbb{F}_q, i = 1, \cdots, \ell, j = 1, \cdots, \ell+2)$;
$\boldsymbol{k}_{\ell+1,\mathsf{del},j}^* := \sum_{i=1}^\ell \alpha_{\mathsf{del},i}\boldsymbol{k}_{\ell,\mathsf{ran},i}^* + \sigma_{\ell+1,\mathsf{del},j}(\sum_{i=1}^n v_{\ell+1,i}\boldsymbol{k}_{\ell,\mathsf{del},i}^*) + \varphi'\boldsymbol{k}_{\ell,\mathsf{del},j}^*$, $(\alpha_{\mathsf{del},i}, \sigma_{\ell+1,\mathsf{del},j}, \varphi' \in_R \mathbb{F}_q, i = 1, \cdots, \ell, j = 1, \cdots, n)$.