Privacy Preserving Disease Treatment & Complication Prediction System (PDTCPS)

Qinghan Xue
Department of CSE
Lehigh University
Bethlehem, PA, USA
qix213@lehigh.edu

Mooi Choo Chuah
Department of CSE
Lehigh University
Bethlehem, PA, USA
mcc7@lehigh.edu

Yingying Chen
Department of ECE
Stevens Institute of Technology
Hoboken, NJ, USA
yingying.chen@stevens.edu

ABSTRACT

Affordable cloud computing technologies allow users to efficiently store, and manage their Personal Health Records (PHRs) and share with their caregivers or physicians. This in turn improves the quality of healthcare services, and lower health care cost. However, serious security and privacy concerns emerge because people upload their personal information and PHRs to the public cloud. Data encryption provides privacy protection of medical information but it is challenging to utilize encrypted data. In this paper, we present a privacy-preserving disease treatment, complication prediction scheme ($PDTCPS$), which allows authorized users to conduct searches for disease diagnosis, personalized treatments, and prediction of potential complications. $PDTCPS$ uses a tree-based structure to boost search efficiency, a wildcard approach to support fuzzy keyword search, and a Bloom-filter to improve search accuracy and storage efficiency. In addition, our design also allows health care providers and the public cloud to collectively generate aggregated training models for disease diagnosis, personalized treatments and complications prediction. Moreover, our design provides query unlinkability and hides both search & access patterns. Finally, our evaluation results using two UCI datasets show that our scheme is more efficient and accurate than two existing schemes.

Keywords

Cloud Computing; PHR; Fuzzy Keyword; Query Privacy; Data Mining

1. INTRODUCTION

In recent years, cloud computing has emerged to be a popular technology that provides scalable and elastic storage and computation resources for enterprises and individuals. More and more organizations and individuals begin to embrace these benefits by outsourcing their data into the cloud [5]. For example: online personal health record (PHR) systems such as Microsoft HealthVaults allow patients to store and manage their own medical records in the public cloud. Such systems allow users easy access and sharing of their personal health data.

Meanwhile, electronic health record systems contain various types of patients' information, which includes their demographics, diagnosis codes, medication, allergies, and laboratory test results [7],[18]. Such health related data is used not just for primary care but also provides a promising avenue for improving healthcare related research. Data consumers can gather large sets of health related information including PHRs from users or healthcare providers and perform large scale analytic tasks, e.g., data-mining tasks to predict disease epidemic [28] or for query and answering [1]. A number of repositories have also been set up to facilitate the dissemination and reuse of patient-specific data for research advancement, e.g., the Database of Genotype and Phenotype (dbGaP) [22]. In addition, work is currently under way to construct the Nationwide Health Information Network (NHIN) [2] to provide privacy-preserving search over distributed, access controlled content.

Although the cloud-assisted healthcare systems offer a great opportunity to improve the quality of healthcare services and potentially reduce healthcare costs, there are many security and privacy concerns. For example, people have started to realize that they would completely lose control over their personal information once it enters the cyberspace. In order to minimize the risk of data leakage to the cloud service providers, sensitive data must be encrypted before being outsourced into the cloud. By doing so, the cloud service providers can only see data in encrypted form and never learn any information about the encrypted data values.

However, this in turn makes data utilization challenging. For instance, it is difficult to apply machine learning techniques to learn from aggregated privately encrypted data for accurate predictions. In order to solve the problem, a set of techniques has been developed (e.g., [6, 13, 16, 19]). While some approaches [13, 16] demonstrated that basic machine learning algorithms such as simple linear classifiers can be performed efficiently to build models over a small scale encrypted dataset, their efficiency degraded rapidly as its size grows. Though other techniques [6, 19] had utilized more sophisticated classifiers (e.g., support vector machine) to address the problem, either they lack security & privacy features or require large computational cost. A recent work [21] designed a scheme which provides machine learning models over encrypted dataset but their encryption scheme has high computational and communication cost. Another re-

ASIA CCS'16, May 30–June 3, 2016, Xi'an, China.
© 2016 ACM. ISBN 978-1-4503-4233-9/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897845.2897893

cent work [10] also designed a scheme which allows data mining over encrypted data but their scheme does not construct encrypted index tree for efficient search. Neither schemes provide features to hide search and access patterns.

To overcome the above limitations, in this paper, we design a privacy-preserving disease treatment, complication prediction scheme ($PDTCPS$), which allows authorized users to conduct searches for disease diagnosis, personalized treatments, and prediction of potential complications of their illnesses. In particular, we design an encrypted index tree which supports fuzzy keyword queries. The tree-based structure is used to provide search efficiency. Each top level node in our encrypted index tree contains an encrypted category keyword that represents a specific body part, e.g., bone & joints, kidneys, etc, and a Bloom filter which contains all the disease keywords classified under this top level node, and their associate fuzzy keyword sets. All relevant information, e.g., associated diseases classified under each top-level node will be stored in the 2^{nd} level nodes. Each 2^{nd} level node (representing k diseases) has three child nodes, one for diagnosis, one for complication prediction and one for treatment options. These three child nodes are leaf nodes. Each leaf node stores relevant information about k diseases, including a training model, its encrypted feature sets and the corresponding Bloom filter containing fuzzy keyword set of each disease. In addition, we include random components in our design to provide query unlinkability, hide search and access patterns. Such features strengthen further the security & privacy capability of our design.

In addition, we present security analysis, and evaluate the effectiveness and efficiency of our proposed scheme using two datasets from the UCI machine learning repository. Our experimental results show that compared to two existing schemes described in [21, 10], our scheme is more efficient (in terms of communication cost) and has higher accuracy than both of these existing schemes. Additionally, our scheme accommodates typos in users' submitted requests, which could not be handled by the existing schemes. In summary, our contributions can be summarized as follows:

- We propose a Privacy-Preserving Disease Treatment, Complication Prediction Scheme ($PDTCPS$), which allows users to conduct privacy-aware searches with high search efficiency and accuracy.
- $PDTCPS$ is designed to handle typos in queries and provide query unlinkability with minimal information leakage.
- Our design allows healthcare providers and the public cloud to collectively generate aggregated training models for disease diagnosis, personalized treatments and prediction of potential illness complications.
- We provide a formal security analysis to justify the privacy-preserving guarantee of our proposed scheme.
- We present simulation results of our proposed scheme using two UCI datasets, namely the PIMA Indians Diabetes and the Breast Cancer Wisconsin Datasets.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides detailed descriptions of the system and threat models, our design goals, and definitions for the techniques used in our design. Section 4 describes our privacy-preserving disease treatment & complication prediction system ($PDTCPS$) in detail. Section 5 presents the security analysis of $PDTCPS$. Section 6 presents the evaluation results where we compare our scheme with two existing schemes using UCI datasets. Section 7 concludes the paper with discussions of our future work.

2. RELATED WORK

2.1 Keyword Search over Encrypted Data

Searchable encryption (SE) is a cryptographic method designed for users to securely conduct keyword search over encrypted data. It was first proposed by Song et al. in [25], where their proposed scheme supports single keyword search without an index and the server must scan the whole document to find the search result. After this work, many improvements and novel schemes have been proposed.

Many SE schemes are either based on public key cryptography (PKC) or symmetric key cryptography (SKC). Boneh et al. [9] proposed the first public key based searchable encryption scheme. Then, to better protect the index and query privacy, Sahai and Waters [24] proposed the idea of Attribute-based Encryption (ABE). In their work, a decrypter could decrypt the message only if his attributes are the same as what is specified by the encrypter. It was later extended to the Key-Policy Attribute-Based Encryption ($KP-ABE$) [15], in which a ciphertext is created with an encryption policy involving a set of attributes. In addition, S.Roy et al. [23] have presented an enhanced Ciphertext Policy Attribute-based Encryption ($CP-ABE$) scheme which provides the user revocation feature. In their scheme, each client's private key is associated with a set of attributes and each ciphertext is encrypted with an access policy so that the encrypted data can only be accessed by the authorized clients.

Other works focus on enriching the search functionality, e.g. providing ranked results, multi-keyword search or fuzzy keyword search. For example, Cao et al. [11] proposed a privacy-preserving multi-keyword ranked search scheme, which allows multi-keyword query and provides similarity ranked results. To improve the search efficiency, Wenhai Sun et al., proposed a tree-based search algorithm in [27]. In addition, Cong Wang et al. [29] proposed a symbol-based tree-traverse searching mechanism to support fuzzy search with constant search time independent of the size of the keyword set. To enrich the search functionality, Chuah et al. [12] introduced a tree structure index to support efficient keyword search and flexible incremental updates.

2.2 Predictive Analysis over Encrypted Medical Data

The increasing availability of commercial options for storing and providing online access to patients' electronic medical records (EMR) has generated much interests among researchers from various fields, e.g., data-mining or bioinformatics to build predictive models using data mining techniques over large scale electronic healthcare data.

Most of these health data research works focus on designing techniques to ensure privacy of sensitive data, or new machine learning classification methods, etc.

2.2.1 *Privacy-preserving Health Data Protection*

Privacy-preserving health data storage is studied by Sun et al. [26], where they design a secure healthcare system $HCPP$ to provide privacy protection on patients' records. Their scheme allows patients to store their encrypted health data on a third-party server and conducts efficient health information retrieval. In addition, in [20], Li et al. have proposed a cloud-based data-sharing framework where the attribute-based encryption (ABE) scheme was used to encrypt patients' medical records. While their scheme provides a strong guarantee that no one could mine any useful information from the encrypted health data, it also makes data utilization a very challenging task. For example, the public cloud can only serve as a remote storage but it cannot conduct data mining over the encrypted data. Moreover, Guo et al. [17] proposed a verifiable privacy-preserving scheme for a cloud-assisted mHealth system which can answer some high level queries but they did not discuss how to generate aggregated training models via machine learning techniques.

2.2.2 *Privacy-preserving Health Data Mining*

Many privacy-preserving data mining schemes for clinical decision support can be grouped into two major categories: randomization based approaches and SMC-based approaches. In the randomization-based approaches, the original data was protected by adding some random noise. In [4], Agrawal and Srikant demonstrated that some statistical properties could still be preserved when adding random noise to the training data. As a result, a Naive Bayes classifier with comparable accuracy could still be obtained from the sanitized data. In [14], Evfimievski et al. presented a new "amplification" method which limits privacy breaches without the knowledge of data distribution. Using their method, randomized data will be added to the original data to avoid privacy breaches. While both proposed randomization methods can protect the sensitive data, a trade off needs to be made between accuracy and privacy.

Secure multiparty computation (SMC) is designed to allow multiple parties, each holding a private input, to collectively perform a computation without disclosing information more than the output reveals. For example, Lin et al. in [21] had designed a cloud-assisted mHealth monitoring system that not only protects the data privacy, but also returns treatment recommendations. In [10] the authors had constructed three major privacy-preserving classifiers including hyperplane decision, Naive Bayes and decision trees to conduct models over encrypted data. While these schemes are secure, they incur large computational and communication costs.

3. PROBLEM FORMULATION

In this section, we first describe our system and threat models. Then, we describe the design goals of our proposed privacy-preserving disease treatments and complications prediction system ($PDTCPS$). Next, we provide descriptions of some important building blocks used in our solution.

3.1 System Model for PDTCPS

PDTCPS consists of four parties: the hospitals, the public cloud server, a fully-trusted authority (TA), and individual clients, as shown in Fig 1.

- Hospitals: Hospitals first collect patients' medical records, encrypt them and store them in the private clouds they owned. The private cloud servers may perform data mining operations over the stored data to generate locally trained models. Based on these models, the hospitals can later diagnose diseases, provide personalized treatments, and predict disease complications for their patients. However, since each hospital may only have limited number of patients associated with a particular disease, the prediction models may not always be comprehensive and accurate. Thus, in order to obtain more accurate prediction, the hospital servers may send relevant information extracted from their trained model securely to the public cloud so that the public cloud can perform data mining operations on the aggregated data received to generate a more accurate prediction model for all participating hospitals to use.
- Semi-trusted public cloud: The public cloud stores the relevant encrypted information sent from each participating hospital, and performs data mining operations to generate predictive models. It also constructs a keyword based encrypted index tree which allows authorized clients to conduct searches based on their individual profiles, lab tests for potential disease diagnosis, treatment options, and risk analysis of potential complications related to their current illnesses.
- Fully-trusted authority (TA): TA is responsible for generating and distributing the symmetric encryption keys to authorized clients and all participating hospitals. It is also responsible for sending relevant disease categorization information, e.g., which disease belongs to which top-level category nodes, to the hospitals and the public cloud.
- Clients: Clients refer to those who wish to conduct searches for disease diagnosis, personalized treatments, and assessing their risks of disease complications caused by their current illnesses.

At the initial phase, TA generates the encryption keys and sends them to all participating hospitals and their authorized clients. Upon receiving the keys, each hospital server first encrypts its data and performs data mining operations to generate locally trained models. Each hospital server then sends the relevant information from the locally trained models securely to the public cloud. The public cloud will then generate aggregated trained models for disease diagnosis, possible treatment models for different groups of patients based on their profiles and medical histories, and prediction models of any potential disease complications. In addition, the public cloud will generate an encrypted index tree which allows clients to search for information more efficiently. Details of what the encrypted index tree contains will be described in Section 4.

When a client wishes to query the public cloud for health related predictions, the client first uses the received keys from the TA to generate a search request and then sends it to the public cloud. After receiving the encrypted query, the public cloud server will perform the search over the encrypted index tree and send back all the relevant answers to the authorized client.

3.2 Adversarial Model

We assume that the trusted authority can be trusted fully and it will not be compromised. As for all participating hospitals, we assume that they are semi-trusted, i.e., they will honestly follow the designated protocols but always curious to gain additional insights from the information sent by other hospitals. They may also collude with the cloud server to find such information.

Similarly, we also adopt a "honest-but-curious" model for the public cloud server as in [30, 29]. Like the hospitals, it will execute the designated protocols honestly but will be

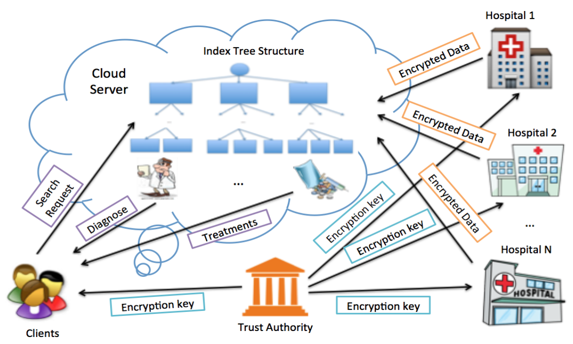


Figure 1: System Model for PDTCPS

curious to infer any extra information it can derive from the information sent by all participating hospitals and from the queries/responses issued/received by the authorized clients.

Depending on the available information to the cloud server, the following two threat models are considered in this work:

- Known Ciphertext Model: The encrypted data, the secure index, encrypted queries and responses are all available to the cloud server.
- Known Background Model: In addition to the available information assumed in the former model, the cloud server can also use statistical information to deduce specific contents in a query. It can even collude with other attackers to derive additional information from the encrypted data.

In addition, we assume users are trusted entities. They obtain authorized keys from the TA.

3.3 Design Goals

To address the security and threat models we have presented earlier, we design a $PDTCPS$ scheme, which allows authorized users to conduct privacy-aware searches for disease diagnosis, personalized treatment and prediction of potential complications based on their individual profiles, laboratory test results, and potential medical histories. Our system is designed with the following goals in mind:

- Fuzzy keyword search: During query generation, an authorized client may make typos while inputting query contents. For example, a client may type “dibetes” instead of “diabetes” in the following query: “(disease=“dibetes”)”. Our scheme should support such fuzzy query and still return relevant information.
- Search Efficiency and Accuracy: The scheme should achieve high search accuracy, i.e., it should return mostly correct answers to search queries. It should also achieve high search efficiency, i.e., the average search time per query is small.
- Privacy Guarantee: Our system should provide privacy guarantees by not leaking sensitive information about stored data or encrypted indices. Our system should provide query privacy and unlinkability. The cloud server should not be able to deduce sensitive contents that have been used for search. Submitted queries should look different each time even if the same keyword and lab results are submitted. Furthermore, the search and access patterns should be hidden from the public cloud server. In other words, the encrypted index structure should be designed such that the server traverses different nodes on the index tree even for the same search request.
- Extensibility: Our system should be designed such that the encrypted index tree as well as trained data models can be updated easily without complete redesign.

3.4 Important Building Blocks

Before we present the detailed description of our newly designed scheme, we first discuss some of the security tools we use in this work, and define a few terminologies.

1. Organization of Information Regarding Various Diseases: Patients may suffer from different types of diseases. To make it easier for $PDTCPS$ we design to answer users' questions regarding diagnosis, treatment options, or potential disease complications, we decide to categorize patients' illnesses similar to how a popular healthcare forum website called patientslikeme organizes different types of diseases. Diseases are categorized based on how they affect human body parts (refer to Fig 2), e.g., Endocrine includes all diseases which affect the endocrine system such as diabetes, hypothyroidism, hyperthyroidism, etc.

For each disease, our system keeps several pieces of important information, namely (i) a trained model for disease diagnosis based on results of laboratory tests, symptoms, (ii)various treatment options based on patients' profiles, and (iii) a trained model for complication prediction based on patients' profiles, laboratory tests, and medical histories, e.g., other diseases a patient may have.

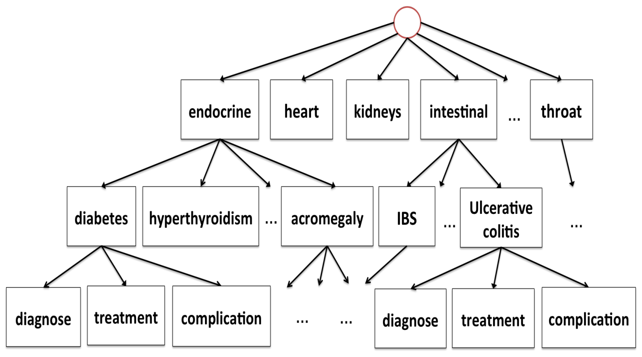


Figure 2: Healthcare Searchable Tree

2. Order-preserving Encryption: Order-preserving symmetric encryption (OPE) is a deterministic encryption scheme which preserves numerical ordering of the plaintexts. It allows order relations between data items to be established based on their encrypted values, without revealing the data itself. For example if $x \leq y$, then $OPE_K(x) \leq OPE_K(y)$, for any secret key K. Thus, with the help of OPE encryption, the server can perform data mining operations over the encrypted data.

3. Parallel SVM method: Support Vector Machines (SVMs) are powerful classification and regression tools, but their computational costs increase rapidly with the size of training instances. Efficient parallel algorithms for constructing SVM models are critical to ensure that SVM can be used for large scale data mining analysis.

The parallel SVM method [31] we use is based on the cascade SVM model where a partial SVM model is constructed for each partition of a large dataset. Then, the partial $SVMs$ are aggregated iteratively as shown in Fig 3. The sets of support vectors from two $SVMs$ are merged into one set and used to create a new SVM. Such a process is repeated until only one set of support vectors remain. This parallel SVM approach allows large scale optimization problems to be divided into smaller independent optimizations.

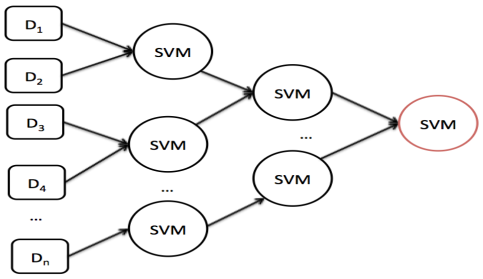


Figure 3: Training process of parallel SVM

4. Parallel Decision Tree method: Decision trees are simple yet effective classification algorithms, but one needs to sort all numerical attributes in order to decide where to split a node within a decision tree, which costs much computation time when a large data set is involved. Thus, it is important to develop parallel version of decision tree algorithms which can be efficient and scalable.

The decision tree method we use is a parallel histogram-based decision tree algorithm for classification [8] where the master node builds the regression trees layer by layer as shown in Fig 4. At each iteration, a new layer is constructed as follows: each node compresses its share of the data using histograms and sends them to the master node. The master node merges the histograms and uses them to approximate the best splits for each leaf node, thereby constructing a new layer. Then, the master node sends this new layer to each participating node, and those nodes construct histograms for this new layer. Therefore, every iteration consists of an updating phase performed simultaneously by all the participating nodes and a merging phase performed by the master node. The communication cost for this method consists of all the histograms sent by the participating nodes to the master and the master sending information of a new layer of the tree to those nodes.

4. PDTCPS SCHEME

As discussed earlier, PDTCPS provides a secure way for clients to diagnose their diseases, predict complications and search for possible treatment options for their illnesses. One important component of our PDTCPS system is the encrypted index tree that the public cloud constructs based on instructions given by the TA. Before we describe our scheme, we first give the definitions of various notations we use.

Notations:

- K_O - the symmetric key for OPE encryption.

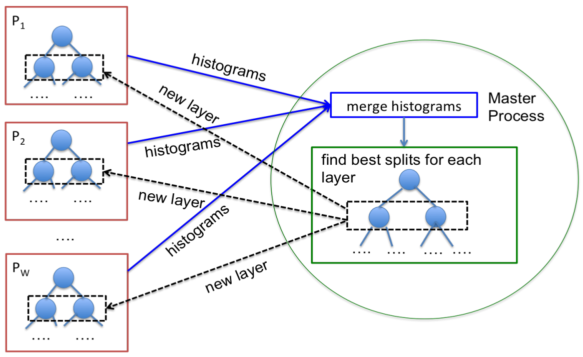
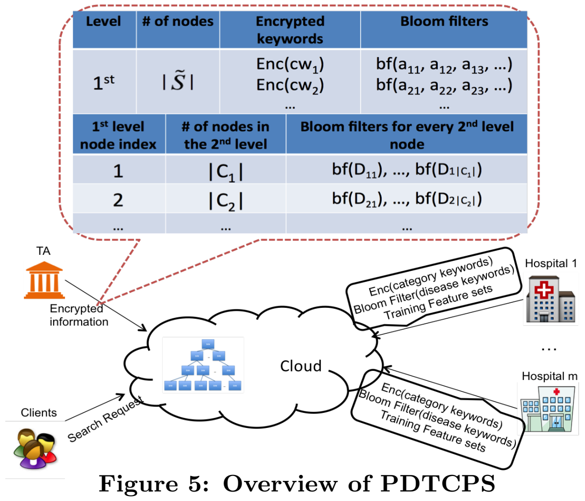


Figure 4: Training process of parallel Decision Tree

- K_B - the Bloom filter generation key.
- K_A - the key used to generate key hash values for keywords, i.e. $Enc(w) = KeyHash(K_A, w)$.
- CW - the category keywords set, denoted as CW=$\{cw_1, cw_2, \cdots, cw_{|CW|}\}$.
- W - the disease keywords set, denoted as W=$\{w_1, w_2, \cdots, w_{|W|}\}$.
- $\widetilde{W}_i$ - a subset of W, indicating the disease keywords in the i^{th} category, denoted as $\widetilde{W}_i$=$\{a_{i1}, a_{i2}, \cdots, a_{i|\widetilde{W}_i|}\}$, where $a_{ij} \in W$.
- $\widetilde{S}$ - a set, indicating the number of children that under each category node, denoted as $\widetilde{S} = \{|C_1|, |C_2|, \cdots |C_{|\widetilde{S}|}|\}$.
- k - the number of diseases stored in every 2nd level node.
- $bf(w_i)$ - a Bloom filter, containing the keyword w_i and its associated fuzzy keywords.
- $\widetilde{sv}$ - a set, indicating the training features of a disease.
- cw_q - the category keyword for the query.
- F - the lab test results set, denoted as $F = \{F_1, F_2, \cdots\}$.
- h - the number of hash functions used in generating the Bloom filter.

4.1 Overview

Fig 5 is an overview of PDTCPS which shows the information provided by the TA, hospitals, and queries submitted by authorized clients.



Level	# of nodes	Encrypted keywords	Bloom filters

1st	$\lvert \widetilde{S} \rvert$	Enc(cw_1)
		Enc(cw_2)
		...
			bf(a_{11}, a_{12}, a_{13}, ...)
			bf(a_{21}, a_{22}, a_{23}, ...)
			...

1st level node index	# of nodes in the 2nd level	Bloom filters for every 2nd level node
1	$\lvert C_1 \rvert$	bf(D_{11}), ..., bf($D_{1\lvert C_1 \rvert}$)
2	$\lvert C_2 \rvert$	bf(D_{21}), ..., bf($D_{2\lvert C_2 \rvert}$)
...	...	...

Figure 5: Overview of PDTCPS

During the encrypted index tree construction phase, the TA sends the public cloud some information to help the public cloud build the encrypted index tree. Specifically, TA sends the public cloud a set of encrypted category keywords, which will form the 1^{st} level nodes. In addition, the TA sends a Bloom filter for each 1^{st} level node which contains keywords of all the illnesses listed under this 1^{st} level node as well as their associated fuzzy keywords. Fuzzy keywords are generated to deal with typos. For example, for a disease or category keyword, w_i = "hypoglycemia", the following wild-card keywords having an edit distance of 1 from the keyword "hypoglycemia": $\{*hypoglycemia$, $h*ypoglycemia$, $hy*poglycemia$, $hyp*oglycemia$, $hypo*glycemia$, $hypog*lycemia$, $hypogl*ycemia$, $hypogly*cemia$, $hypoglyc*emia$, $hypoglyce*mia$, $hypoglycem*ia$, $hypoglycemi*a$, $\cdots\}$ are inserted into the Bloom filter. Note that, it is easy to extend our system to support multiple edit distances. (e.g., generate one Bloom filter per edit distance).

The TA also sends information regarding the number of children each 1st level node will have, e.g., top-level node i will have $|C_i|$ 2nd level nodes. Each 2nd level node has a Bloom filter containing k encrypted disease keywords and their associated fuzzy keyword sets (to address typos). All Bloom filters associated with 2nd level nodes are also sent to the public cloud. The public cloud then stores those Bloom filters in the appropriate child nodes of first level nodes.

Each 2nd level node represents k diseases and has three child nodes, namely (i) diagnosis, (ii) complication prediction, and (iii) treatment options. These child nodes are leaf nodes. The diagnosis node will contain the training models for disease diagnosis of the k diseases that this 2nd level node represents. Each training model has an associated disease token (which is a secure Bloom filter that contains hash values of a disease keyword with its typos) for the public cloud to determine which training model to use when it processes a query. Similarly, the complication prediction node contains k training models, each for predicting potential complications which may arise of a particular disease. Finally, the treatment node contains training models for k diseases, one for each illness. Each training model represents an aggregated model constructed by the public cloud using encrypted information sent by each hospital. The training model is built using patients' profiles, disease treatments, laboratory tests, etc, and is used to assess the best treatment option for a particular patient based on his personal profile, and/or laboratory test results.

As for the clients, they may use their personal profiles and lab tests results to generate search requests. After receiving an encrypted search request, the public cloud server first finds a matched category in the 1st level category nodes. Then, the server will only search for matching results among the child nodes of that best matched 1st level category node. This can significantly reduce the search time because the server merely searches information within this relevant subtree structure, only a subset of the whole information collection. The server goes through the child nodes of this selected category node to find $k = 2$ best matched 2nd level nodes. Next, based on the query identifier, the server randomly selects one of the k matched level 2 nodes, and traverses into its sub-tree structure based on the query type, e.g., diagnosis, complication or treatment. After finding the matched leaf node, the cloud server will return the answers using the appropriate training model for that query. For example, based on the disease token and query type, the cloud server selects the appropriate training model to see if a client has suffered this disease or predict potential complications that may arise or the treatment options for this particular disease based on that client's unique profiles and laboratory test results.

4.2 Detail Design of PDTCPS

We present more detailed descriptions of the proposed scheme in this section.

4.2.1 Index Tree Construction

The public cloud constructs a keyword based encrypted index tree which allows authorized clients to conduct searches for health related questions based on their individual profiles and lab tests results.

Here, we describe how the encrypted index tree is constructed. In our design, we use $SHA-256$ as our keyed hash function, and use L-bit Bloom filters to handle typos.

1. Operations performed by the TA:

(i) In the initialization phase, a secret key $SK = \{K_O, K_B, K_A\}$ is produced by the trust authority where (a) K_O is a symmetric key for OPE operation; (b)K_B is the generation key for the Bloom filter generation; (c) K_A is the key used for computing key hash values of category keywords.

(ii) Then, TA generates a set of key hash values of category keywords, $Enc(CW) = \{Enc(cw_1),\ Enc(cw_2),\ \cdots\}$ which will form the 1st level nodes.

(iii) For every category i, TA also produces a set: $\widetilde{W_i} = \{a_{i1},\ a_{i2},\ \cdots,\ a_{i|\widetilde{W_i}|}\}$, where a_{ij} is a disease keyword belonging to category i. Next, for each keyword a_{ij}, the TA generates a fuzzy keyword set: $\{a_{ij_1}, a_{ij_2}, \cdots\}$, where a_{ij_z} is a single-typo keyword of a_{ij}. The TA inserts the keyed hash values of all relevant disease keywords and their associated fuzzy keyword sets into a L-bit Bloom filter, $bf(\widetilde{W_i})$, using the secret key K_B.

(iv) In addition, TA determines the number of children nodes, $|C_i|$ for each category node i and forms the set $\widetilde{S} = \{|C_1|, |C_2|, \cdots, |C_{|\widetilde{S}|}|\}$. Then, for each child node (e.g., the j^{th} child node of the i^{th} category), it stores a keyword set D_{ij}, which contains k disease keywords. Next, TA generates a Bloom filter $bf(D_{ij})$, which contains those k keywords as well as their associated fuzzy keywords. Our solution inserts the same disease into k different 2nd level nodes so that the cloud server can go through k different nodes (based on query identifiers) to find a matched leaf node even with the same keyword search request. Thus, both the search and path patterns can be hidden from the cloud server.

(v) Finally, TA delivers all the generated ciphertexts including $\{Enc(CW), \widetilde{S}, \{BFD(\widetilde{W_1}), \cdots, BFD(\widetilde{W_{|S|}})\}\}$, where $BFD(\widetilde{W_i}) = \{I_{cw_i}, bf(\widetilde{W_i}), \{bf(D_{i1}), bf(D_{i2}), \cdots, bf(D_{i|C_i|})\}\}$ and I_{cw_i} is a category index, to the cloud server.

(vi) It also sends both encrypted category keywords, $Enc(CW)$ and the secret key SK to every hospital. The secret key SK is also sent to all authorized clients.

2. Operations performed by hospitals:

(i) Every hospital H_m contains a category set $\widetilde{CW_m} = \{kw_1, kw_2, \cdots, kw_{|\widetilde{CW_m}|}\}$ and a disease keyword set $\{G_m(kw_1),\ G_m(kw_2),\ \cdots,\ G_m(kw_{|\widetilde{CW_m}|})\}$, where $G_m(kw_i) = \{b_{i1},\ b_{i2},\ \cdots,\ b_{i|G_m(kw_i)|}\}$ and b_{ij} is a disease keyword.

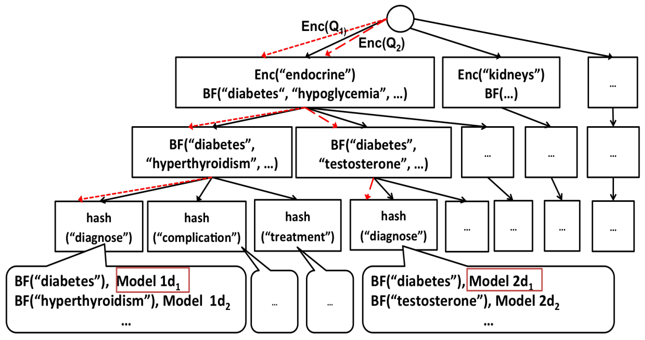
(ii) Then, hospital H_m generates $Enc(CW_m)$ which consists of all key hash values of category keywords in CW_m.

(iii) For each illness b_{ij} which H_m has relevant patients' information, it also generates $bf(b_{ij})$ using the secret key K_B.

(iv) Next, H_m uses a classification method to extract the training feature set for that illness b_{ij}, denoted as $\widetilde{sv}(b_{ij})$. Later, it encrypts this training feature set using OPE and the K_O key to produce $\widetilde{SV}(b_{ij}) = OPE_{K_O}(\widetilde{sv}(b_{ij})) + r_{ij}$, where r_{ij} are some random value sets. The random values are added to ensure the participating hospitals cannot uncover the true values of these feature vectors each hospital sends even if some hospitals collude with the cloud server.

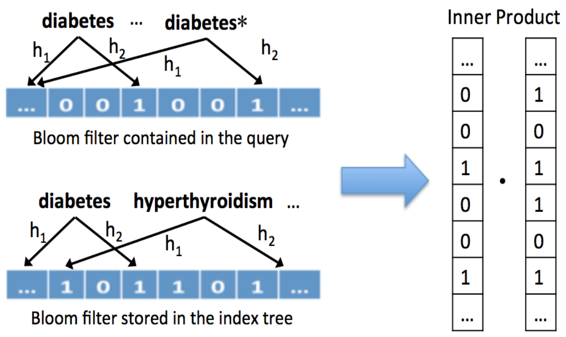
(v) Finally, hospital H_m sends $\{Enc(\widetilde{CW_m}), \{BSV(G_m(kw_1)),\ \cdots,\ BSV(G_m(kw_{|\widetilde{CW_m}|}))\}\}$, where $BSV(G_m(kw_i)) = \{I_{kw_i},\ \{bf(b_{i1}),\ \widetilde{SV}(b_{i1})\},\ \cdots,\ \{bf(b_{i|G_m(kw_i)|}),\ \widetilde{SV}(b_{i|G_m(kw_i)|})\}\}$ and I_{kw_i} is the category index for category keyword kw_i, to the cloud server.

3. GenIndex: (i) The cloud server first builds the 1st level



Enc(Q1)
Enc(Q2)
Enc("endocrine") BF("diabetes", "hypoglycemia", ...)
Enc("kidneys") BF(...)
BF("diabetes", "hyperthyroidism", ...)
BF("diabetes", "testosterone", ...)
hash ("diagnose")
hash ("complication")
hash ("treatment")
hash ("diagnose")
BF("diabetes"), Model 1d1
BF("hyperthyroidism"), Model 1d2
BF("diabetes"), Model 2d1
BF("testosterone"), Model 2d2

Figure 6: PDTCPS Index Tree Structure



diabetes
diabetes*
h1
h2
Bloom filter contained in the query
diabetes
hyperthyroidism
Bloom filter stored in the index tree
Inner Product

Figure 7: Inner Product Computation

nodes, where the i^{th} category node stores the encrypted keyword $Enc(cw_i)$ and the corresponding Bloom filter $bf(\widetilde{W_i})$.

(ii) Then, the public cloud uses the received information to build the 2^{nd} level, where the j^{th} second level node of the i^{th} first level node stores the corresponding Bloom filter $bf(D_{ij})$.

(iii) For each 2^{nd} level node, the cloud server constructs three children nodes, one for each query type, i.e., "diagnose", "complication" and "treatment". An integer value can be used to represent each query type.

(iv) For each received set of information from a hospital, the cloud server first uses the received $Enc(kw_i)$ to find the matched 1^{st} level category node. Then, it computes the inner product values between each of the received $bf(b_{ij})$ and the Bloom filters stored in the $|C_i|$ child nodes under the i^{th} category node to find the k matched 2^{nd} level nodes. Next, the cloud server traverses into their leaf nodes to find the right training model.

(v) If no training model exists, the newly received training model will be stored. If a training model already exists, the public cloud generates a new model by combining previously stored feature vectors for that disease with the most recently received feature vectors to generate a new training model.

With the procedures outline above, the cloud server finally constructs the encrypted index tree, which is shown in Fig 6.

Query Generation

To provide query unlinkability, we need to generate a different search request even for the same keyword query.

(i) Given a query Q=$\{cw_q, (F_1, \cdots, F_i, \cdots), x_q, t_q\}$, where cw_q is the category keyword, F_i is either a personal attribute of a client or his lab test result i, x_q is the disease keyword and t_q is the query type.

(ii) An authorized client first generates a random query id ID_q and a keyed hash value of the category keyword as $Enc(cw_q)$.

(iii) Each F_i will be encrypted as $OPE_{K_O}(F_i) + R_i$ using the received encryption key K_O and a random value R_i. This step ensures that the same F_i results in different encrypted value and hence provides query unlinkability.

(iv) The client also generates a fuzzy keyword set: $\{x_{q1}, \cdots, x_{q_i}, \cdots\}$, where x_{q_i} is a single-typo keyword of x_q. Then, the client generates $bf(x_q)$ using the secret key K_B.

(v) Finally, the encrypted search request $Enc_{SK}(Q)$=$\{ID_q, (OPE_{K_O}(F_1) + R_1, \cdots, OPE_{K_O}(F_i) + R_i, \cdots), Enc(cw_q), bf(x_q), hash(t_q)\}$, is submitted to the cloud server.

Search Process

(i) Upon receiving the search request $Enc_{SK}(Q)$, the server first checks if $Enc(cw_q)$ can be matched with the stored encrypted keywords $Enc(CW)$ in the 1^{st} level nodes.

(ii) If it is not found, then the cloud server computes the inner products of Bloom filter $bf(x_q)$ in the query with $|\widetilde{S}|$ Bloom filters stored in the 1^{st} level nodes. The one with the best match will be the selected 1^{st} level node. (Fig 7)

(iii) Next, the cloud server searches through the child nodes of this selected 1^{st} level category node by performing the following operations:

• Compute the inner product values between the $bf(x_q)$ and the stored $|C_i|$ Bloom filters in the 2^{nd} level nodes.

• Find the top k nodes among those $|C_i|$ nodes in the second level.

• Next, select one of k matched 2^{nd} level nodes, node j, using $j = ID_q \ mod \ (k)$ and travels into its sub-tree nodes based on the query type.

(iv) After finding the matched leaf node, the cloud server can find the appropriate training model to diagnose disease, predict potential complications or determine the best treatment options for a client based on his query type, t_q.

5. SECURITY ANALYSIS

In this section, we analyze the privacy characteristic of $PDTCPS$ against possible attacks by various entities involved in our system. Adversaries in our system could be participating hospitals, network eavesdroppers, or even the cloud server. For instance, hospitals and the cloud server in our system are assumed to be semi-trusted, implying that they follow the protocol execution, but may attempt to learn additional information. A network eavesdropper could have the resources to monitor all messages in the network or the messages sent by a particular hospital or a client.

5.1 Network Eavesdroppers

For network eavesdroppers, $PDTCPS$ achieves privacy preserving mainly via encrypted communication. Our designed scheme guarantees that attackers cannot uncover any knowledge of any content within the ciphertext as long as eavesdroppers cannot obtain the cryptographic keys. In addition, by adding randomness in each encrypted query, the adversaries cannot conduct frequency analysis to gain additional information about submitted queries and hence no sensitive information is revealed.

5.2 Semi-honest Hospitals

In the presence of semi-honest hospitals, our scheme achieves information-theoretic security. Specifically, our design adds some randomness to the encrypted training features generated by each hospital before they are being sent to the cloud server. Thus, participating hospitals cannot gain additional information on the ciphertext sent by other hospitals even if they know the secret key.

5.3 Semi-honest Cloud Server

In this subsection, we will show how $PDTCPS$ satisfies several search privacy requirements:

• Index and Query confidentiality under both the known ciphertext model and the known background model. More details are provided in subsequent subsections.

• Query unlinkability: Our $PDTCPS$ generates different search requests even with the same query keyword and hence provides query unlinkability to a certain extent.

• Hiding access pattern: Our design ensures that the cloud server traverses different nodes to find a match even with the same keyword search request and hence the access patterns are hidden from the server.

5.3.1 Security Analysis of PDTCPS Under the Known Ciphertext Model

Here, we adapt the simulation-based security model in [27] to prove that our scheme can be secure under the known ciphertext attack. Before proving, we first introduce some notations that will be used in the proving process.

• History: It is an index set I and a query set $Q = \{Q_1, Q_2, \cdots\}$, denoted as $H = (I, Q)$.

• View: The cloud server can only see the encrypted form of a H, denoted as $VI(H)$, including the secure indexes $Enc(I)$ and the encrypted search requests $Enc(Q) = \{Enc(Q_1), Enc(Q_2), \cdots\}$.

• Trace: A trace is a set of queries, denoted as $Tr(H) = \{Tr(Q_1), Tr(Q_2), ...\}$. $Tr(Q_i)$ captures the information for each query Q_i including the search pattern PA_{Q_i}, and the outcome of the search RE_{Q_i} which is available to the cloud server to gain additional information.

As in [27], our proof is based on the following argument: given two histories that produce the same trace, if the cloud server cannot distinguish which history is produced by the simulator, then the cloud server cannot learn additional knowledge beyond the information that the system is willing to leak.

We adopt a simulator that can simulate a view $VI(H)'$ indistinguishable from cloud server's view $VI(H)$. The simulator works as follows:

1. For the encrypted query $Enc(Q_1)$, the simulator generates $Enc(Q_1)'$ as follows:

(i) The simulator first selects random strings $\{s_1, s_2, s_3\}$, where $s_i \in \{0,1\}^U$, and then sets $ID'_{Q_1} = s_1$, $hash(cw'_{Q_1}) = s_2$ and $hash(t'_{Q_1}) = s_3$ separately. U is the length of hash-value, ID'_{Q_1} is the query identifier, and cw'_{Q_1} is the category keyword in that query and t'_{Q_1} is its query type.

(ii) The simulator also generates a L'-bit vector $v \in \{0,1\}^{L'}$ and sets $bf(x'_{Q_1}) = v$ where x'_{Q_1} mimics the disease keyword in the query.

(iii) Next, the simulator builds a vector which represents encrypted attributes used in the query, $Enc(F'_{Q_1}) = \{G_1, G_2, \cdots\}$, where G_i is a random string chosen from $\{0,1\}^U$.

(iv) After the above steps, the following encrypted query, $Enc(Q_1)'$=$\{ID'_{Q_1}$,$Enc(F'_{Q_1})$,$hash(cw'_{Q_1})$,$bf(x'_{Q_1})$,$hash(t'_{Q_1})\}$, is simulated.

2. Based on the search pattern PA_{Q_1}, the simulator can generate the $Enc(I)'$ as follows:

(i) Let us assume PA_{Q_1} goes through category node $ca(1)$, intermediate node $im(1)$ and leaf node $ln(1)$ of the index tree.

(ii) The simulator first sets the $Enc(ca(1))' = hash(cw'_{Q_1})$.

(iii) Then, the simulator adds $bf(x'_{Q_1})$ to $bf(ca(1))'$.

(iv) The simulator also sets $bf(\widetilde{D'}_{Q_1})' = bf(\widetilde{D'}_{Q_1})' + bf(x'_{Q_1})$, where $|\widetilde{D'}_{Q_1}| = k$ and $\widetilde{D'}_{Q_1}[(ID'_{Q_1})mode(\text{k})] = im(1)$.

3. For subsequent queries such as Q_j with search pattern PA_{Q_j} which goes through category node $ca(j)$, intermediate node $im(j)$ and leaf node $ln(j)$ of the index tree where $2 \leq$ i $\leq j \leq |Q|$, the simulator does the following:

(i) If $ca(j)$, $im(j)$, $ln(j)$ are not same as $ca(i)$, $im(i)$, $ln(i)$, then the simulator repeats the same process as simulating $Enc(Q_1)'$ and $Enc(I)'$.

(ii) If $ca(j)$ is the same as $ca(i)$ but $im(j) \neq im(i)$, then the simulator repeats the same process as simulating $Enc(Q_1)'$ and $Enc(I)'$ with the condition that $hash(cw'_{Q_j})$=$hash(cw'_{Q_i})$.

(iii)If $im(j)$ is the same as $im(i)$ but $ln(j) \neq ln(i)$, then the simulator sets $hash(t'_{Q_j}) \neq hash(t'_{Q_i})$ and also generates all the other necessary information.

(iv) If the search pattern PA_{Q_j} ends at the same leaf node $ln(i)$ as the previous query Q_i, then the simulator sets $hash(t'_{Q_j}) = hash(t'_{Q_i})$ and does the following:

• If the search result RE_{Q_j} for the query Q_j, is not the same as RE_{Q_i}, then the simulator repeats the same process as simulating $Enc(Q_1)'$ and $Enc(I)'$ with the condition that the $Enc(F'_{Q_j})$ is different from the $Enc(F'_{Q_i})$.

• If the search result is the same, then the simulator sets $bf(x'_{Q_j}) = bf(x'_{Q_i})$ and generates the $Enc(F'_{Q_j})$, which is similar to the $Enc(F'_{Q_i})$.

4. After all the queries have been simulated, the simulator does the following:

• It converts each $bf(ca(i))'$ and $bf(im(i))'$ into L'-bit $\{0,1\}^{L'}$ vectors by replacing the elements bigger than 1 with 1.

• It adds $Enc(F'_{Q_i})$ to the training feature set $\tilde{sv}(x'_{Q_i})$ to make sure that the query result is the same as RE_{Q_i}. Note that the training feature set $\tilde{sv}(x'_{Q_i})$ is attached to the appropriate simulated leaf node.

5. The simulator outputs the view $VI(H)'$=($Enc(I)'$, $Enc(Q)'$).

In summary, the $Enc(I)'$ and $Enc(Q)'$ can generate the same trace as the one that the cloud server has. Thus, we claim that no probabilistic polynomial-time (P.P.T) adversary can distinguish between the view $VI(H)'$ and $VI(H)$.

5.3.2 Security Analysis of PDTCPS Under the Known Background Model

In this subsection, we analyze the security of $PDTCPS$ under the known background attack model. For each query Q_i we generate the encrypted search request as follows: $Enc(Q_i)$=$\{ID_{Q_i}$,$Enc(F_{Q_i})$,$hash(cw_{Q_i})$,$bf(x_{Q_i})$,$hash(t_{Q_i})\}$. Since a random value set is introduced during the query generation, $PDTCPS$ produces different search requests even for the same query. Thus, our scheme can achieve query unlinkability such that it makes it hard for the cloud server to link one transformed request to another even if both contain the same keyword.

In addition, since in the known background model, the cloud server can deduce the statistical information by analyzing the search and path patterns for each query. Thus, it is important to hide those information from the cloud. In our scheme, we have extended every 2^{nd} level node to contain k different keywords so that the cloud server randomly selects one of the k matched nodes containing the desired keyword. Therefore, both the search and path patterns can be hidden from the cloud server.

Discussion: Since OPE is a deterministic encryption, so it is subjected to two known security vulnerabilities, namely (i) frequency-based attack where adversaries use frequency distributions of ciphertext and plaintext to infer their correspondence and (ii) order-relations among plaintexts where attackers can easily break the encryption via sorting of known values of plaintexts and ciphertexts using domain knowledge. However, since we have added random values in $PDTCPS$, attackers simply cannot infer such information. Thus, our design is safe against both frequency and domain attacks.

6. PERFORMANCE EVALUATION

We implemented PDTCPS and conduct our experiments on a Mac Pro with an Intel Core i5 processor running at 2.6GHz and 8GB memory. The following performance metrics are used to evaluate our scheme ($PDTCPS$) and two other proposed solutions, namely the CAM [21], and the hyperplane decision-tree based scheme ($HDBS$) [10]:

- Index construction time, which is the time incurred in generating the proposed index tree structure;
- The generation time, testing time, accuracy, communication and storage costs of the training model;
- The accuracy of the search results.

First, we select $|\widetilde{S}|$ categories based on the major categories in Medical Health provided in the Patientslikeme website [1], (e.g., Endocrine, Intestinal, Throat etc), as the 1^{st} level nodes of the index tree.

Then, based on these $|\widetilde{S}|$ categories, we extract $\sum_{i=1}^{|\widetilde{S}|} |C_i|$ unique disease keywords, e.g., Endocrine includes all diseases which affect the endocrine system such as diabetes, hypothyroidism, hyperthyroidism and so on. Next, we map all these $\sum_{i=1}^{|\widetilde{S}|} |C_i|$ distinct keywords into their appropriate categories and build the encrypted index tree, where each leaf node represents k disease keywords. We also set k=2, the length of the Bloom filter, L, to $64bytes$, and use h=2 hash functions to insert keywords and their associated fuzzy keyword sets to a Bloom filter in our $PDTCPS$ scheme.

6.1 Construction and Communication Costs For Index Tree

The index construction process contains two major steps:

- TA generates sets of encrypted information including the encrypted category keywords, Bloom filters, and the number of children that under each category node. Then, it sends these Bloom filters to the public cloud.
- After receiving the information above, the public cloud stores all these information in the index tree.

This index construction cost is only a one-time computation cost. Since the encrypted index tree contains $|\widetilde{S}|$ category nodes and $\sum_{i=1}^{|\widetilde{S}|} |C_i|$ second level nodes, the TA needs to generate $|\widetilde{S}|$ encrypted category keywords and $|\widetilde{S}|+\sum_{i=1}^{|\widetilde{S}|} |C_i|$ Bloom filters(BFs). Fig 8(a) shows the generation cost for a L-bit Bloom filter and from the results we can see the generation cost increases linearly with the number of inserted keywords. In addition, it needs to send all these $|\widetilde{S}|+\sum_{i=1}^{|\widetilde{S}|} |C_i|$ Bloom filters to the public cloud to be stored in the encrypted index tree. Since the results in our system show a linear relationship between the time and the number of disease keywords, so the realistic overhead of our system will increase linearly according to the number of disease keywords. For example, base on the Dewey Decimal system, which is a library classification system, we can further cluster all the existing 30,000 human diseases into almost 60 categories. Assuming $|\widetilde{S}|$=60 and each top-level node has 500 child nodes, then the total computational cost can be computed as follows: it takes 0.39 ms to insert 60 fuzzy keywords into the BF of a child node and 0.39ms*500*31/60=101 ms to insert 500*31 fuzzy keywords into the BF of each category node. Thus, the overall index construction time for 30,000 diseases is 18 sec. Furthermore, to ensure the collision rate of BF at each category node to 1%, we need to use a BF of length 305 Kbytes. In addition, each child node contains 2 disease keywords where the average keyword length is 15. Thus, we need a 1.2 Kbytes Bloom filter for each child node to ensure a 1% collision rate. Therefore, the total one-time communication cost that TA incurs to send relevant information to the cloud for index tree construction of 30,000 diseases is (305*60+1.2*500*60)=53 Mbytes.

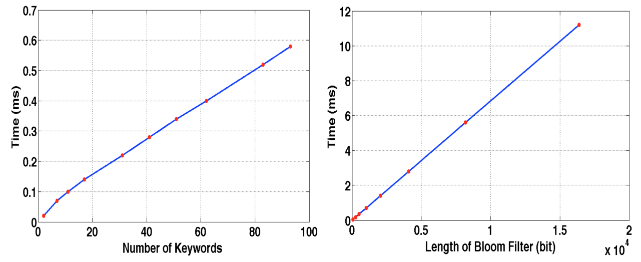


Figure 8: (a)Bloom Filter Generation Cost Vs Number of Keywords. (b)Inner Product Computation Cost Vs Bloom Filter Length .

6.2 Training Model Evaluation

1. Training Model Generation Method:

As for the training model generation cost, we first describe the construction processes for $PDTCPS$, CAM and $HDBS$ schemes. Since we do not have access to any dataset for complications and treatments, here we only evaluate the performance of our model for disease diagnosis.

(i) $PDTCPS$: For each disease associated with a leaf node, the cloud server generates a training model based on the received training feature sets from all hospitals. Here, we use the parallel SVM method described in Section 3 to construct an aggregate SVM model. By having each hospital conducts its own data mining and sends only encrypted support vectors makes our solution more efficient and scalable.

(ii) CAM [21]: This scheme uses a parallel histogram-based decision tree algorithm to generate the training model where every iteration consists of an updating phase performed simultaneously by all the hospitals and a merging phase performed by the cloud server. At each iteration, a new layer is constructed as follows: each hospital compresses its share of the data using histograms and sends them to the cloud. Then, the cloud server merges the histograms and determines the best splits for each node in the decision tree, thereby constructing a new layer. Next, it sends this new layer to each hospital, and the hospitals construct histograms for this new layer. Finally, the cloud server can build the regression tree layer by layer through the iterations.

(iii) Hyperplane Decision Based Scheme ($HDBS$) [10]: This scheme introduces a sophisticated approach to perform machine learning on encrypted data. All hospitals send their encrypted datasets to the public cloud server. The public cloud server generates an aggregated training model based on all these encrypted datasets using homomorphic encryption method. Next, the client generates an encrypted search and submits to the public cloud. The public cloud traverses the encrypted index tree as described before and sends relevant answers back to the client in encrypted form. The client then decrypts the returned response to obtain the answer.

2. Training Model Performance:

In this sub-section, we conduct Exp 1 and Exp 2 to evaluate the performance of the above three schemes.

(A) Exp 1: In the experiment, we use the Pima Indians Diabetes Data Set from the UCI machine learning repository [3], which contains 768 instances with 9 attributes of two labeled classes. We first select 90% of the Pima dataset as training set, S_1, and the remaining 10% as the test set T_1. Then, based on the distribution (e.g., mean or standard deviation) of each attribute, we generate two synthetic datasets from S_1 denoted as S_{11} and S_{12}, where S_{11} contains 1384 instances, and S_{12} contains 4152 instances. Next, we partition the synthetic datasets as follows (i) we partition each synthetic dataset into m equal parts and assign each part to one hospital. (ii) Since in the real world different hospital may have different data size, so we also divide each synthetic dataset into m unequal parts, and assign each of them to one hospital.

(B) Exp 2: To ensure that the conclusions we draw from Exp 1 is reliable, we also use the Breast Cancer Wisconsin (Original) Data Set, which contains 699 instances with 10 attributes and two class labels, to conduct Exp 2. The same method used in Exp 1 is used to generate the dataset for each hospital.

After data generation, the hospitals then extract the training features from their assigned datasets and encrypt them using the OPE algorithm, where the encryption complexity is largely based on the bit length of each feature. For example, our experiments show that using only the 1^{st} 10 bits of the encrypted value produce similar prediction accuracy in disease prediction. The OPE algorithm takes 7.1ms to encrypt a 10-bit length feature. Thus, we only use the first 10 bits of the encryption value for all our experiments to reduce encryption time without affecting accuracy.

(C) Performance Evaluation: Tables I and II show the evaluation results for all the above three schemes. Note that (i) the reported storage cost is the cost of storing one training model for a particular disease, and (ii) no $HDBS$ result is reported for the diabetic dataset because we have no access to their codes, and they did not have published results using that dataset. One can see that the training time for our scheme ($PDTCPS$) is much smaller than the CAM and $HDBS$ schemes described in [21, 10]. The CAM scheme is inefficient since it needs multiple interactions between hospitals and cloud server to generate the aggregated decision-tree, which greatly increases the training cost. $HDBS$ uses the aggregated dataset for SVM training while $PDTCPS$ uses parallel SVM for training, hence $HDBS$ incurs more training time than $PDTCPS$.

Tables I and II also show that our training model generation process incurs smaller communication cost than the CAM and $HDBS$ schemes.$PDTCPS$ incurs the least cost because the hospitals only need to send the encrypted training features instead of all the instances to the cloud server. Whereas in the CAM scheme, the communication cost is largely due to the histograms that are sent by the hospitals. Meanwhile, in order to transform the ciphertext from one encryption form into another, the $HDBS$ scheme requires multiple interactions between a client and server, which incurs much communication cost. The tables also show that by using SVM rather than decision tree, $PDTCPS$ achieves higher accuracy than CAM.

In addition, from Tables I and II, we can see that the CAM scheme incurs less test time than $PDTCPS$ when the dataset size is small. This is expected because the test time for the CAM scheme is largely based on the height of the decision tree. Thus, when the dataset is small, the height of the decision tree is also small, which leads to low test evaluation cost. Whereas in $PDTCPS$, the number of instances in the dataset has little impact on the test evaluation cost since it only depends on the number of attributes. $PDTCPS$ only incurs about 0.035Kbytes for Exp1 and 0.04Kbytes for Exp2 to store a training model. However, to increase the efficiency for future training model updates, we may also store the encrypted training feature sets. The reported storage cost for $PDTCPS$ in Tables 1&2 shows the storage cost incurred when such feature sets are also stored.

6.3 Search Evaluation

In this sub-section, we evaluate the search performance of $PDTCPS$.

1. Search Over Encrypted Index: The search operation executed at the cloud server side consists of the inner product calculation for the nodes contained in the index tree. If a node contains the keyword(s) in the query, the corresponding bits in both Bloom filters will be 1 thus the inner product will return a high value. Figure 8 (b) shows that the inner-product computation time grows linearly with the length of the Bloom filter. This is intuitive because the cloud server needs to go over all the bits in Bloom filters before computing the final inner product values. Assuming that there are $|\tilde{S}| = 60$ categories, and each category has 500 diseases, then on the average, a query without an encrypted category keyword needs to search through 30 top level category nodes and 250 2nd-level nodes, then the average search time will be (30*1.7+250*0.0067)=53s since each inner product computation takes 6.7 ms with $L = 1.2Kbytes$ and 1.7s with $L = 305Kbytes$. However, the search time is only about (250*0.0067)=1.675s with an encrypted category keyword included in the query. The search time for queries without category keywords can be reduced by using the bed-tree structure [12] to create more hierarchy for category keywords so that fewer category nodes need to be searched.

2. Search Accuracy: In our experiment, we adopt the widely used performance metric, namely false positive, denoted as FP, to measure the search result accuracy. The false positive rate of a L-bit Bloom filter with h hash functions can be computed as $(1 - \frac{1}{L})^{nh}$, where n is the number of keywords inserted into that Bloom filter. The number of the inserted keywords in a Bloom filter for a disease keyword can be computed as $n = 2 * l_i + 1$, where l_i is the number of characters of that disease keyword w_i.

Figure 9(a) shows how the false positive rate of our scheme varies as the number of inserted keywords changes when $L = 1.2Kbytes$. One observation is that the false positive is very low when l_i is small, i.e. 0.6% at $l_i = 15$ which is the average character length of our disease keywords.

Figure 9 (b) shows the performance of our scheme when the length of a Bloom filter is varied. Although large Bloom

Table 1: **Training Model Evaluation for Exp1**

Scheme	Number of Attributes	Number of Instances	Equal Data Size	Number of Leaf Nodes	Training Time	Communication Cost	Testing Time	Accuracy	Storage Cost
PDTCPS	9	1384	Yes		0.05s	8.09KB	0.024ms	81.6%	7.9KB
			No		0.05s	8.1KB	0.024ms	80.3%	7.7KB
		4152	Yes		0.13s	23.48KB	0.024ms	76.3%	21.8KB
			No		0.15s	23.50KB	0.024ms	80.3%	21.9KB
CAM	9	1384	Yes	48	2.0s	8.7KB	0.015ms	75.0%	0.18KB
			No	41	1.8s	7.9KB	0.14ms	75.0%	0.17KB
		4152	Yes	187	7.8s	28.1KB	0.026ms	68.4%	0.6KB
			No	178	6.9s	24.60KB	0.024ms	69.7%	0.58KB

Table 2: **Training Model Evaluation for Exp2**

Scheme	Number of Attributes	Number of Instances	Equal Data Size	Number of Leaf Nodes	Training Time	Communication Cost	Testing Time	Accuracy	Storage Cost
PDTCPS	10	1680	Yes		0.004s	1.01KB	0.027ms	92.9%	1.1KB
			No		0.005s	1.07KB	0.027ms	92.1%	1.2KB
		5040	Yes		0.011s	1.67KB	0.027ms	92.1%	1.5KB
			No		0.012s	1.72KB	0.027ms	90.6%	1.5KB
CAM	10	1680	Yes	24	1.5s	9.6KB	0.010ms	86.4%	0.06KB
			No	20	1.3s	8.7KB	0.09ms	88.5%	0.05KB
		5040	Yes	42	5.8s	20.8KB	0.015ms	84.3%	0.1KB
			No	36	5.5s	18.9KB	0.013ms	86.4%	0.09KB
HDBS	10	699			0.032s	35.84KB	151.1ms		

^{1}The results of the *HDBS* scheme are extracted from [10] and scaled to the same CPU environment used to evaluate our scheme;

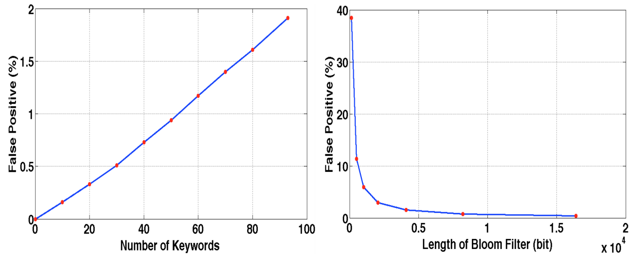


Figure 9: False Positive Rate of the Bloom filter. (a) With varying numbers of keywords. (b) With various Bloom filter lengths.

filter can better reduce the false positive rate, it may increase both the search time and the storage cost (since the cloud server needs to store these Bloom filters in the index tree). Thus, there is a trade-off among the false positive rate, search time, and the storage cost of our scheme. For example, we can tune the parameters, i.e. L, l_i, to specifically fit a particular accuracy and storage requirements.

Therefore, the total accuracy can be further computed by combining both false positives in the searching model and error rates in the prediction results as follows:

$$Acc_{total} = 1 - (P_f + (1 - P_f) * C_f) \qquad (1)$$

where P_f is the false positive for the searching model and C_f is the error rate for the training model. Typically, the Bloom filters are designed to achieve a $P_f = 1\%$ and C_f as shown in Tables 1 & 2 ranges from 76.3 to 92.9%.

7. CONCLUSION

In this paper, we have proposed a Privacy-Preserving Disease Treatment, Complication Prediction Scheme (*PDTCPS*), which allows users to conduct privacy-aware searches for health related questions based on their individual profiles and lab tests results. Our design also allows healthcare providers and the public cloud to collectively generate aggregated training models to diagnose diseases, predict complications and offer possible treatment options. In addition, to enrich search functionality and protect the clients' privacy, our scheme can support fuzzy keyword search and query unlinkability. Moreover, *PDTCPS* also hides access patterns and hence addresses the security threat via exposed access patterns identified in previous searchable encryption schemes. Finally, we validate the practicality of our scheme by evaluating our scheme using two *UCI* datasets. The results show that *PDTCPS* is secure against different adversarial situations, and has better performance than two existing schemes.

In the near future, we would like to enhance our scheme to support more complex query types (e.g., range queries), while preserving the privacy of the query keywords. For example, a medical researcher may want to find the number of diabetic patients who have taken a specific drug for a long time, and yet still suffer a high blood sugar level, by submitting a query like "(50<age<80) AND (sex="female") AND (illness="diabetes") AND (drug="humira") AND (duration>5 years) AND (blood-sugar>7%)". In addition, we would like to improve the performance of our scheme using other types of encrypted health data. For example, we may include disease diagnosis using *MRI* images, and hence enhance our scheme to deal with encrypted image features.

ACKNOWLEDGEMENT

This work was supported by the US National Science Foundation awards CNS-1217379, CNS-1217387 and CNS-0954020.

8. REFERENCES

[1] Patientslikeme, https://www.patientslikeme.com/.
[2] NHIN, http://www.hhs.gov/healthit/healthnetwork.
[3] UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets.html.

[4] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM Sigmod Record*, volume 29, pages 439–450. ACM, 2000.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. In *Communications of the ACM*, pages 50–58, April 2010.

[6] M. Barni, P. Failla, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Privacy-preserving ecg classification with branching programs and neural networks. pages 452–468. IEEE, 2011.

[7] J. L. N. B.B. Dean, J. Lam, Q. Butler, D. Aguilar, and R. J. Nordyle. Use of electronic medical records for health outcomes research: a literature review. In *Medical Care Research Review*, 2010.

[8] Y. Ben-Haim and E. Tom-Tov. A streaming parallel decision tree algorithm. volume 11, pages 849–872. JMLR. org, 2010.

[9] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[10] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. *Crypto ePrint Archive*, 2014.

[11] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. volume 25, pages 222–233. IEEE, 2014.

[12] M. Chuah and W. Hu. Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data. In *ICDCSW*, pages 273–281, 2011.

[13] W. Du, Y. S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM*, pages 222–233. SIAM, 2004.

[14] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222. ACM, 2003.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[16] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *Information Security and Cryptology–ICISC 2012*, pages 1–21. Springer, 2013.

[17] L. Guo, Y. Fang, M. Li, and P. Li. Verifiable privacy-preserving monitoring for cloud-assisted mhealth systems. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1026–1034. IEEE, 2015.

[18] E. Lau, F. S. Mowat, M. A. Kelsh, J. C. Legg, N. M. Engel-Nitz, H. N. Watson, and et al. Use of electronic medical records (emr) for oncology outcomes research:assessing the comparability of emr information to patient registry and health claims data. In *Clinical Epidemiology*, 2011.

[19] S. Laur, H. Lipmaa, and T. Mielikäinen. Cryptographically private support vector machines. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–624. ACM, 2006.

[20] 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. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):131–143, 2013.

[21] H. Lin, J. Shao, C. Zhang, and Y. Fang. Cam: cloud-assisted privacy preserving mobile health monitoring. volume 8, pages 985–997. IEEE, 2013.

[22] M. Mailman, M. Feolo, Y. Jin, M. Kimura, K. Tryka, R. Bagoutdinov, and et al. The ncbi dbgap database of genotypes and phenotypes. In *National Genetology*, 2007.

[23] S. Roy and M. Chuah. Secure data retrieval based on ciphertext policy attribute-based encryption (cp-abe) system for the dtns. Technical report, Citeseer, 2009.

[24] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005*, pages 457–473. Springer, 2005.

[25] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[26] J. Sun, X. Zhu, C. Zhang, and Y. Fang. Hcpp: Cryptography based secure ehr system for patient privacy and emergency healthcare. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 373–382. IEEE, 2011.

[27] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 71–82. ACM, 2013.

[28] M. J. Tildesley, T. A. House, M. C. Bruhn, R. J. Curry, M. ONeil, J. E. Allpress, and et al. Impact of spatial clustering on disease transmission and optimal control. In *Proceedings of National Academy Science*, 2010.

[29] C. Wang, K. Ren, S. Yu, and K. M. R. Urs. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *INFOCOM, 2012 Proceedings IEEE*, pages 451–459. IEEE, 2012.

[30] 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 *INFOCOM, 2014 Proceedings IEEE*, pages 2130–2138. IEEE, 2014.

[31] K. Xu, C. Wen, Q. Yuan, X. He, and J. Tie. A mapreduce based parallel svm for email classification. volume 9, pages 1640–1647, 2014.