Secure and Privacy-Preserving Querying of Personal Health Records in the Cloud

Samira Barouti, Feras Aljumah, Dima Alhadidi, Mourad Debbabi

Secure and Privacy-Preserving Querying of Personal Health Records in the Cloud

Samira Barouti, Feras Aljumah, Dima Alhadidi, and Mourad Debbabi

Concordia Institute for Information Systems Engineering (CIISE)
Concordia University, Montreal, Canada

Abstract. Personal Health Records (PHR) are user-friendly, online solutions that give patients a way of managing their own health information. Many of the current PHR systems allow storage providers to access patients' data. Recently, architectures of storing PHRs in cloud have been proposed. However, privacy remains a major issue for patients. Consequently, it is a promising method to encrypt PHRs before outsourcing. Encrypting PHRs prevents health organizations from analyzing medical data. In this paper, we propose a protocol that would allow health organizations to produce statistical information about encrypted PHRs stored in the cloud. The protocol depends on two threshold homomorphic cryptosystems: Goldwasser-Micali (GM) and Paillier. It executes queries on Kd-trees that are constructed from encrypted health records. It also prevents patients from inferring what health organizations are concerned about. We experimentally evaluate the performance of the proposed protocol and report on the results of implementation.

1 Introduction

Electronic health records are usually managed by different healthcare providers including primary care physicians, therapists, hospitals and pharmacies. Consequently, it is difficult to get a single patients history due to the fact that it is spread between multiple providers. It has become a recent trend for patients to take these matters into their own hands by managing their own records using a Personal Health Record (PHR) system. PHRs systems allow patients to manage their medical data, giving them the ability to create, view, edit, or share their medical records with other users in the system as well as with healthcare providers [1]. In the past few years, many providers have created platforms to manage PHRs with features including flexible access control, mobile access, and complex automated diagnoses that analyze PHRs and alert patients when a preventive checkup is needed. These providers include Microsoft HealthVault [2] and Dossia [3]. Due to the sensitivity nature of health data, security concerns have prevented many patients from using PHR systems. Many of the providers of the current PHR systems have the ability to access all patient records.

Recently, architectures for storing PHRs in the cloud have been proposed [1]. However, this does not solve the privacy problem and the latter remains an

issue for many patients. Since these records are stored on cloud servers, it means that these servers have the ability to read any medical record in the system. In addition, if an attacker is able to compromise a cloud server, then all the PHRs would be exposed. For these reasons, researchers have begun searching for a way to allow patients storing their medical records in the cloud using a Database-as-a-Service (DaaS) model while preserving their privacy. DaaS is a category of cloud computing services that enables IT providers to deliver database functionality as a service. Encrypting PHRs before outsourcing appears to be a promising solution in this domain. However, it prevents health organizations from analyzing medical data for research purposes. To better understand what caused a disease, health organizations and researchers need as much data as possible about the infected patients.

In this paper, we propose a protocol that allows health organizations producing statistical information about encrypted PHRs stored in the cloud. The proposed protocol also does not enable patients to infer about what health organizations are concerned about; not to worry or panic patients targeted by the queries. Intuitively, the proposed protocol works as follows: Patients are organized in small groups. The patients of a given group jointly generate public keys for encryption. They later encrypt their PHRs using their public keys and send the ciphertext to the cloud server. Encrypted records are stored in Kd-trees constructed by the cloud server for each group. To execute SQL queries, Kd-trees are traversed in the cloud server. Finally, the cloud server aggregates the results and sends the final query result to the health organization. However, realizing this seemingly simple system presents a number of significant challenges. First, the search should be performed on encrypted records. To achieve this, our proposed protocol depends on the homomorphic properties of two semantically-secure encryption schemes. Using homomorphic schemes, specific operations can be performed on the encrypted records directly without the need for decryption. More specifically, query predicates are evaluated using the Goldwasser-Micali (GM) cryptosystem [4] and Fischlins protocol [5] whereas query aggregate functions are computed using Paillier cryptosystem [6]. Second, the engagement of the patients in the protocol execution should be minimal. We achieve this by using threshold cryptosystems such that the decryption process is performed by a specific number of patients, namely the threshold k.

The contributions of our paper can be summarized as follows:

- We propose a protocol, which allows health organizations producing statistical information about PHRs stored in the cloud and encrypted using semantically-secure encryption schemes. The main characteristics of the protocol are the following:
  - It preserves the privacy of health organizations and patients.
  - It supports aggregate queries such *count*, *sum*, *max* and *min*.
- We design and implement a prototype of the proposed protocol and we also report on the experimental results.

The rest of the paper is organized as follows. Section 2 discusses the execution environment. Section 3 briefly overviews the literature that the proposed solu-

tion depends on. Section 4 presents a protocol to find the maximum/minimum value of encrypted inputs without resorting to deterministic encryption schemes. In Section 5, the proposed protocol is presented. The security and complexity analysis of the protocol as well as the experimental results are discussed in Section 6. Section 7 presents the related work. Finally, concluding remarks as well as a discussion of future work are presented in Section 8.

2 Execution Environment

In this section, we first identify the involved entities. Then, we present the assumptions underlying the system design.

2.1 Entities

There are three main entities: (1) *Patients* who own health records and want to store them on a cloud server while keeping them confidential from the cloud server and health organizations. A common way to protect health records stored on cloud servers is through encryption, (2) *Cloud server* that stores the encrypted health records of the patients and executes the queries of the health organization over the encrypted records. The cloud server will assign an assisting server to each group. The assisting server is a cloud computing instance, which will be responsible for storing the encrypted records of the patients and executing the SQL queries of the health organization, (3) *Health organization* that execute queries over the encrypted records of the patients and produce statistical information.

2.2 Assumptions

We assume that there is no fully trusted entity in the environment and all entities are semi-honest. Semi-honest adversaries follow the protocol steps, but they try to extract information about other entities' input or output. This is a common security assumption used in secure multiparty computation literature [7] and it is realistic in our problem scenario since different organizations are collaborating for mutual benefits. Hence, it is reasonable to assume that parties will not

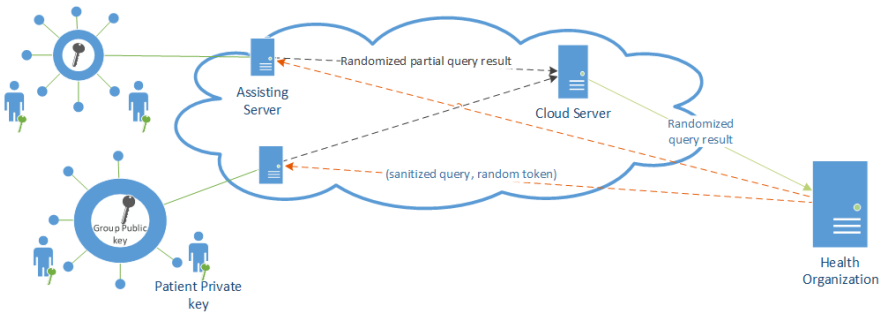


Fig. 1: System Architecture

deviate from the defined protocol. However, they may be curious to learn additional information from the messages they receive during protocol execution. The cloud server and the assisting servers are modeled as "honest but curious" in the sense that they will execute requests correctly, but they are not reliable to maintain data confidentiality. Regarding query privacy, we assume that the shape of SQL queries submitted by health organizations is public whereas the constants contained in the query predicates are private [8, 9]. We also assume that there is no collusion between the different parties and that there are mechanisms that ensure integrity and availability of the remotely stored data. Our scheme focuses only on confidentiality and privacy issues and does not provide protection against attacks such as data tampering and denial of service.

3 Building Blocks

Homomorphic Encryption: It is a form of encryption where a specific algebraic operation performed on the plaintext is equivalent to another (possibly different) algebraic operation performed on the ciphertext. The Paillier's scheme [6] is an additive homomorphic public key encryption. Using Paillier's scheme, given two ciphertexts $E(x)$ and $E(y)$, an encryption of their sum $E(x+y)$ can be efficiently computed by multiplying the ciphertexts modulo a public key N, i.e., $E(x+y) = E(x).E(y) \bmod N$. The Goldwasser-Micali (GM) cryptosystem is a semantically-secure scheme based on the quadratic residuosity problem. It has XOR homomorphic properties, in the sense that $E(b) \,.\, E(b') = E(b \oplus b') \bmod N$ where b and b' are bits and N is the public key. Variations of the homomorphic Paillier and GM cryptosystems are the distributed threshold decryption schemes in which the decryption is performed by a group of participants, rather than one party [10, 11]. In this case, each participant would obtain a share of the secret key by executing the distributed key generation algorithm detailed in [12].

Private Comparison: Yao's classical millionaires' problem [13] involves two millionaires who wish to know who is richer. However, they do not want to find out inadvertently any additional information about each other's wealth. More formally, given two input values x and y, which are held as private inputs by two parties respectively, the problem is to securely evaluate the Greater Than (GT) condition $x > y$ without exposing inputs. In this paper, we employ Fischlin's protocol [5] for the private comparison because it allows us to compare two ciphertexts encrypted with the GM crytosystem using the same public key. Fischlin's protocol takes as input two ciphertexts encrypted using GM cryptosystem and produces ciphertext sequences, namely Δ and c that are encrypted by the same public key. The decryption of these sequences would reveal the result of comparing the private inputs without revealing anything beyond the result of the comparison. Fischlin's protocol utilizes the XOR-homomorphic GM cryptosystem to privately compute:

$$\begin{aligned} x > y &\iff \bigvee_{i=1}^{n} \left(x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} (x_j = y_j) \right) \\ &\iff \bigoplus_{i=1}^{n} \left(x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} \neg (x_i \oplus y_i) \right) \end{aligned}$$

where $|x| = |y| = n$.

Kd-trees: Kd-trees [14] are binary trees where the keys differs between levels. Kd-trees are used extensively in searching multidimensional data, in particular database records. The normal way of constructing a balanced Kd-tree is to sort the records according to the i-th attribute and split them into left and right parts, with respect to the median element. The process is then repeated recursively on both parts taking into consideration the $(i+1)$th attribute.

4 Secure Maximum/Minimum Computation

As a major step in our proposed protocol, we need to execute the max/min aggregate queries over encrypted records. In this section, we provide a cloud-based solution that calculates the maximum/minimum of encrypted values owned by some parties. Formally, given n inputs $v_1, \ldots, v_n$ owned by the parties $P_1, \ldots, P_n$ respectively, the cloud server wishes to securely compute $max(v_1, v_2, \ldots, v_n)$ and $min(v_1, v_2, \ldots, v_n)$. We assume that the parties are not malicious and they correctly carry out the prescribed steps. The proposed cloud-based solution relies on Fischlin's protocol [5] and the threshold GM cryptosystem [10]. We explain the technique to find the max but it can be easily modified to find the min.

Parties jointly generate the public key pk for k-out-of-n threshold GM cryptosystem such that at least k patients are required to fully decrypt a ciphertext [10]. Parties then encrypt their values and outsource them to the cloud server. The cloud server initializes the current maximum to the encryption of a small negative value with the threshold GM cryptosystem. Afterwards, the cloud compares the current maximum with the encrypted value of the party P_i using Fischlin's protocol. The outputs of Fischlin's protocol are sequences of λ elements. To decide whether the encrypted value of P_i is greater than the current maximum (If there exists a sequence of λ quadratic residues), the cloud server contacts k other parties and sends the generated sequences for them. Each party performs calculation on the sequences using her share of the factors of the public key [10] using the threshold GM decryption. The results are then submitted to the cloud server. Afterwards, the cloud server combines the results received from k parties and decide if the encrypted value of P_i is greater than the current maximum based on the quadratic residuosity of the combinations. If the output indicates that the current maximum is greater, there is no need to update the current maximum. Otherwise, the cloud server sets the current maximum to the value of P_i and repeats the same process with P_{i+1}. After comparing the current maximum with all the existing values, the cloud server sends the encrypted maximum value to k members for decryption. This idea can be extended to enable a cloud server to sort the encrypted values without knowing the secret key and the plaintexts. In this case, any comparison-based sorting algorithm can be utilized and the comparison is performed on the encrypted values by exploiting Fischlin's protocol.

5 Secure Execution of Health Queries in Cloud

In this section, we present a protocol that enables health organizations to produce statistical information about encrypted personal health records stored in the cloud server. The proposed protocol prevents patients from inferring what health organizations are concerned about. The health organization's input is an aggregate SQL query that consists of exact-matching and interval-matching predicates over multiple attributes combined with logical operators (AND/OR/NOT). The cloud server's input is the encrypted health records of the patients. The naive approach to achieve these objectives is that the health organization communicates with each patient and securely evaluates queries on her record. This can be achieved by exploiting Fischlin's protocol for private comparison. However, this approach incurs excessive communication and computation overhead on the health organization side that is linear to the number of patients.

To reduce this overhead, patients are organized into smaller groups. The patients in each group jointly generate two public keys for Goldwasser-Micali [4] and Paillier [6] encryption schemes. Then, they encrypt their records and outsource them to the cloud server for storage. The cloud server assigns an assisting server to each group. Assisting servers are responsible for securely executing health organization's queries on the database of each group to obtain partial results. Assisting servers then collaborate to combine the partial results into the query result and report it to the health organization. In the following, we elaborate the basic steps of our protocol that protects the data privacy of patients and the query privacy of the health organization.

5.1 Key Generation and Tree Construction

Assuming the total number of N patients, the cloud server defines $L = \lfloor\sqrt{N}\rfloor$ groups. It then randomly maps and assigns each patient into exactly one group. Let $n = \lceil\frac{N}{L}\rceil$ denotes the number of patients in each group. The cloud server assigns an assisting server to each group, which is responsible for executing health organizations' queries over the medical database of patients. Assisting servers collaborate with each other to obtain the query result from the partial results and send it to the health organization. In the i-th group, the patients execute the distributed key generation algorithms for the threshold Paillier and the threshold GM cryptosystems to obtain the public keys pk'_i and pk_i for Paillier and GM cryptosystems. We utilize the protocols explained in [10] and [15] for the threshold GM and the threshold Paillier cryptosystems, respectively. These protocols depend on distributed RSA key-generation protocols [16, 12] without the need to a trusted dealer. Following the execution of the key-generation protocols, each patient obtains a single public key and a share of the secret key. The threshold cryptosystems enable the patients to encrypt their record with a single public key while at least a minimal number of patients are required to decrypt a ciphertext.

Example 1. Consider health records with the attributes *Age* and *Surgery*, where the value of *Surgery* specifies the type of the surgery that a patient undergoes

(e.g. 1: Transgender, 2: Plastic, 3:Vascular, 4: Urology). The total number of patients is $N = 10$; therefore, these patients are organized in $L = \lfloor\sqrt{10}\rfloor = 3$ groups, namely, G_1, G_2 and G_3. Assume that patients 1,9 and 10 are assigned to G_1; patients 2, 4, 5 and 8 to G_2 and patients 3, 6 and 7 to G_3, randomly. Furthermore, the patients in the group G_i jointly generate the public key pk_i and pk_i' for the GM and the Paillier cryptosystems, respectively. The members of G_i encrypt their records with pk_i and pk_i' as presented in Fig. 2. The encrypted tables are then outsourced to the cloud server.

To store the shares of a secret key, we assume the secret key is stored on secure hardware such as FPGA [17]. These devices are designed in such a way that after a patient places a key into the on-board key memory on the device, it cannot be read externally. After the secret key of each patient and the group public keys (for Paillier and GM cryptosystems) have been written, the FPGA can be delivered to the cloud operator for installation.

	Age	**Surgery**
Patient 1	34	1
Patient 2	39	2
Patient 3	20	1
Patient 4	59	3
Patient 5	63	4
Patient 6	27	2
Patient 7	78	4
Patient 8	11	2
Patient 9	83	3
Patient 10	42	3

Table 1: Health Records

	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_P$	**Surgery**$_P$
Patient 1	$E_{pk_1}(34)$	$E_{pk_1}(1)$	$E_{pk_1'}(34)$	$E_{pk_1'}(1)$
Patient 9	$E_{pk_1}(83)$	$E_{pk_1}(3)$	$E_{pk_1'}(83)$	$E_{pk_1'}(3)$
Patient 10	$E_{pk_1}(42)$	$E_{pk_1}(3)$	$E_{pk_1'}(42)$	$E_{pk_1'}(3)$

(a) G_1 Data set

	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_P$	**Surgery**$_P$
Patient 2	$E_{pk_2}(39)$	$E_{pk_2}(2)$	$E_{pk_2'}(39)$	$E_{pk_2'}(2)$
Patient 4	$E_{pk_2}(59)$	$E_{pk_2}(3)$	$E_{pk_2'}(59)$	$E_{pk_2'}(3)$
Patient 5	$E_{pk_2}(63)$	$E_{pk_2}(4)$	$E_{pk_2'}(63)$	$E_{pk_2'}(4)$
Patient 8	$E_{pk_2}(11)$	$E_{pk_2}(2)$	$E_{pk_2'}(11)$	$E_{pk_2'}(2)$

(b) G_2 Data set

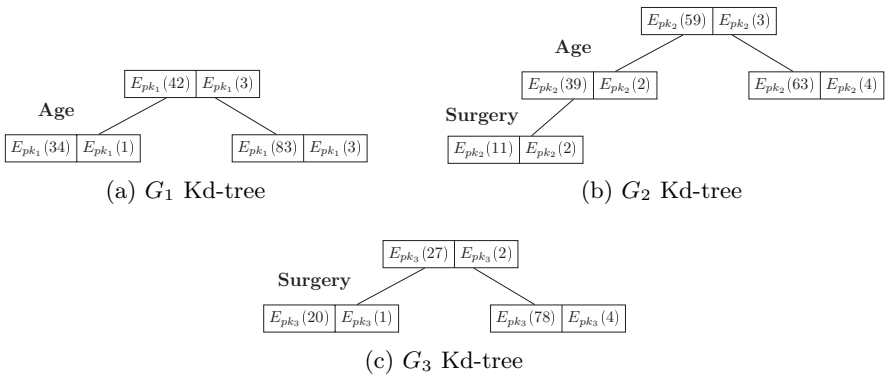
	Age$_{GM}$	**Surgery**$_{GM}$	**Age**$_P$	**Surgery**$_P$
Patient 3	$E_{pk_3}(20)$	$E_{pk_3}(1)$	$E_{pk_3'}(20)$	$E_{pk_3'}(1)$
Patient 6	$E_{pk_3}(27)$	$E_{pk_3}(2)$	$E_{pk_3'}(27)$	$E_{pk_3'}(2)$
Patient 7	$E_{pk_3}(78)$	$E_{pk_3}(4)$	$E_{pk_3'}(78)$	$E_{pk_3'}(4)$

(c) G_3 Data set

Fig. 2: Outsourced Health Records in Groups

The patients encrypt each record using both Paillier and GM cryptosystems by the group public keys. Therefore, the encrypted record of each patient has two columns for each attribute in the database: one column that contains the encryption of the attribute value using the group public key for the threshold Paillier cryptosystem, and another column that stores the GM encryption of the attribute value using the group public key for the threshold GM cryptosystem. Finally, the cloud server assigns an assisting server to each group. Each assisting server collects the encrypted health records and organizes them as a Kd-tree, as described in Section 3.

Example 2. (Continued from Example 1) The generated Kd-trees for each group are shown in Fig. 3. The partitioning attributes in each group may be different.



(a) G_1 Kd-tree (b) G_2 Kd-tree

(c) G_3 Kd-tree

Fig. 3: Generated Kd-trees

5.2 Query Sanitization and Token Generation

The health organization wishes to execute an SQL query such that the constants in the predicates are not revealed to the patients and the cloud server. Therefore, the health organization sanitizes the query by replacing the constants contained in the predicates by their GM encryption using the public key of each group. Furthermore, the health organization uses a token for each group that is encrypted by the group's public key. This token can be manipulated by assisting servers to produce a noisy query result. Generating the token depends on the type of the aggregate function. For *count* and *sum*, the health organization generates L random numbers $R_1, R_2, \ldots, R_L$ such that $R = R_1 + R_2 + \ldots + R_L$. The random share R_i will be the token that is sent to the assisting server of the i-th group. For the *max* and *min*, the health organization generates a random number R as the token for all groups. The health organization then encrypts the token of each group by the Paillier cryptosystem using the group public key. The health organization forwards the sanitized query together with the encrypted token to assisting servers. Therefore, in this step the health organization should create L sanitized queries and L encrypted tokens.

Example 3. Suppose that the health organization's query is:

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE Surgery} = 1$$

The sanitized query that will be forwarded to the i-th group would be

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE Surgery} = E_{pk_i}(1)$$

In addition, since the function is *max*, the health organization generates a random number R and uses it as the token for all groups. The health organization then encrypts the token using the Paillier encryption by the public key of each group and forwards the encrypted token together with the sanitized query to the corresponding assisting server.

5.3 Tree Traversal and Query Execution

To execute the health organization's query, the assisting servers must traverse the Kd-trees, constructed from the encrypted records of the patients. To do so, the assisting servers follow the tree traversal algorithm. The search begins from the root; the assisting server uses Fischlin's protocol and the threshold GM decryption to evaluate the query predicate on the root record. Based on the result of the query evaluation, the search is continued on the left tree or the right subtree or both. At the end of this step, the assisting servers will end up with the records that satisfy the query predicate. The assisting servers then compute the encrypted query result depending on the type of the aggregate function as follows:

- *count*: The assisting server of the group G_i counts the number of records that are reported as the query result and encrypts this value using the Paillier cryptosystem with the group public key.
- *sum*: The assisting servers encrypt 0 (as the current sum) by the Paillier encryption using the group public key. While traversing the tree, if at each level the conditions in the query predicate are satisfied, the assisting server projects the record over the Paillier-encrypted column targeted by the aggregate function and multiplies it by the the current sum to update the query result. At the end, the assisting server will end up with the sum that is encrypted with the group public key using the threshold Paillier cryptosystem.
- *max, min*: Initially, the assisting servers pick up a small negative number (or a large positive in case of *min*) that denotes the current *max/min* value and encrypts it by the GM and the Paillier cryptosystems using the group public keys. GM-encrypted ciphertext is utilized for the comparison while Paillier-encrypted ciphertext is used for generating noisy query result. During the tree traversal if a record satisfies the query condition(s), the assisting server projects the record over the columns that contain the GM- and the Paillier-encryption of the record. It then executes Fischlin's protocol using the encrypted current *max/min* value and the GM-encrypted value, to find out if this record has greater (resp. smaller) value or not. If so, the assisting server initializes the current *max/min* value to the GM- and the Paillier-encrypted ciphertexts. Otherwise, the current *max/min* value remains unchanged. At the end, the assisting servers end up with the query result encrypted with the Paillier and GM cryptosystems. For the remaining step of the protocol, the assisting servers only need the Paillier-encrypted ciphertext.

At the end of this step, the assisting servers obtain the partial query result (that has been encrypted using the Paillier scheme by the public key of the group).

Example 4. (Continued from Example 3) Considering the Kd-trees presented in Fig. 3 and the sanitized query

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE Surgery} = E_{pk_i}(1)$$

All assisting servers receive an encrypted token $E_{pk'_i}(R)$ from the health organization. The assisting server of the group G_i extracts $E_{pk_i}(1)$ from the query

and performs the point search on the Kd-tree constructed by the patients in the group G_i. The assisting servers report the records that satisfy the predicate $\mathsf{Surgery} = E_{pk_i}(1)$ by executing Fischlin's protocol and the threshold GM cryptosystem. The record of Patient 1 in G_1 and the record of Patient 3 in G_3 satisfy the predicate. Therefore, the output of the tree traversal for the assisting servers of the groups G_1, G_2 and G_3 will be $\{E_{pk_1}(34), E_{pk'_1}(34)\}$, $\{E_{pk_2}(-1000), E_{pk'_2}(-1000)\}$ and $\{E_{pk_3}(20), E_{pk'_3}(20)\}$, respectively. The resulted outputs will be projected over the column $\mathbf{Age}_P$ to obtain $\{E_{pk'_1}(34)\}$, $\{E_{pk'_2}(-1000)\}$ and $\{E_{pk'_3}(20)\}$ as the encrypted query result.

5.4 Query Result Decryption

So far, the assisting servers have obtained the encrypted partial query result, which we will call group results from now on. Therefore, the assisting servers must collaborate with each other to compute the final query result and submit it to the health organization. The group results are encrypted with different keys. Therefore, in order to compute the final query result, the group results must be in plaintext. The assisting servers first obfuscate the group results because they are not willing to reveal these results to each other. The obfuscation is performed by the mean of multiplying the group result by the encrypted token, sent by the health organization (both of them are ciphertexts generated by the same key under the Paillier cryptosystem). The obfuscation allows the assisting servers to collaborate with each other to calculate the noisy query result while hiding the actual group result. In addition, since the noise is generated by the health organization, it allows the health organization to recover the actual query result from the noisy query result. Afterwards, each assisting server decrypts the noisy group result that is encrypted by the Paillier cryptosystem by contacting patients in its group. The assisting servers then need to obtain the final noisy query result by aggregating their group results. In this context, the assisting servers send the noisy group results in plaintext to the cloud server. The cloud server then aggregates the partial noisy query results to obtain the noisy query result. In the case of *count* and *sum*, the cloud server adds up all the group results and submits the summation to the health organization. In the case of max and min aggregate functions, the cloud server executes the *maximum/minimum* algorithm on the plaintexts and sends the resulted value to the health organization. Notice that the noise generated for obfuscating the *maximum/minimum* of all groups is the same, therefore it will not affect the algorithm correctness (i.e., if $a < b$ then $a + R < b + R$). Finally, the health organization in its turn subtracts the noise and obtains the query result.

Example 5. (Continued from Example 4) We have seen that the result of executing the SQL query

$$\texttt{SELECT MAX(Age) FROM } D \texttt{ WHERE } \mathsf{Surgery} = 1$$

on the groups G_1, G_2 and G_3 was $\{E_{pk'_1}(34)\}$, $\{E_{pk'_2}(-1000)\}$ and $\{E_{pk'_3}(20)\}$ respectively. Moreover, the token sent by the health organization to the i-th group is $E_{pk'_i}(R)$. The assisting servers multiply the received token $E^R_{pk'_i}$ by all

records in the encrypted query result to obtain the encrypted noisy query result, i.e., $\{E_{pk'_1}(34 + R)\}$, $\{E_{pk'_2}(-1000 + R)\}$ and $\{E_{pk'_3}(20 + R)\}$. The assisting servers then decrypt these ciphertexts to obtain $34 + R$, $-1000 + R$ and $20 + R$. They send their noisy plaintexts to the cloud server. The cloud server executes the maximum algorithm on the $34 + R$, $-1000 + R$ and $20 + R$ and eventually ends up with $34 + R$ as the maximum. The cloud server forwards $34 + R$ to the health organization. The health organization subtracts the noise R to obtain 34 as the result of executing the SQL query on the medical database.

6 Security & Efficiency Analysis

6.1 Security Analysis

Assuming the semi-honest adversary model and no collusion between the patients, the security of the protocol depends on the steps where the parties exchange information and it is conducted as follows:

- *Health organization-Cloud server*: The health organization sends the sanitized query and the token that are encrypted by the semantically-secure encryption schemes using patients' public keys. Therefore, the cloud server is not able to decrypt it [4, 6].
- *Patient-Cloud server*. The patients sends their medical records, encrypted using semantically-secure encryption schemes that are secure against semi-honest adversary [4, 6].
- *Cloud server-Patient*. The cloud server communicates with the patients in order to execute Fischlin's protocol, that is proven to be secure in the presence of semi-honest adversaries [5, 10, 11].
- *Assisting servers*. The interactions between assisting servers are required to aggregate the noisy group results and acquire the randomized final query result. Since the query results have been randomized by a number that is generated by the health organization, the assisting servers are not able to extract the actual query results from the noisy results.

Moreover, the output of each subprotocol is the input to another subprototcol. Therefore, according to the Composition Theorem [18], the entire protocol is secure. The main concern with threshold cryptosystems comes from the collusion attack. We address in the following the possible attacks resulted from the collusion between the different parties. The threshold decryption will be compromised if the number of colluding clients under the control of an adversary exceeds the threshold k. Any collusion that contains less than k patients in each group cannot learn any information about the ciphertext sequences Δ and c, generated for comparison as well as constants in the query of the health organization. The most serious collusion attacks are when: (i) the cloud server colludes with more than k patients in each group to recover the encrypted database records, (ii) the cloud server and at least k patients in any group collude to infer constants in the query. In practice, we can increase the threshold k such that attackers will not be able to compromise too many patients. Despite simplicity, this mechanism has

Health Organization		Assisting Servers	
Communication	Computation	Communication	Computation
$O(\sqrt{N})$	$O(k\sqrt{N})$	$O(kT(N))$	$O(kT(N))$

Table 2: Communication and Computation Cost

two disadvantages: First, the number of the required online patients is increased. Second, the communication cost on the assisting servers is increased because they need to communicate with more parties for decryption. Therefore, there should be a trade-off between availability/efficiency and security by choosing a proper value for k.

It should be noted that the proposed protocol does not protect the privacy of patients from being identified through the query result. There is a significant body of works on distributed privacy preserving data mining (e.g. constructing decision trees [19] and differential privacy [20–22]) that provide a rich and useful set of tools to protect the record owner (i.e., patients) from being identified through query results. They allow a trusted server to release obfuscated answers to aggregate queries to avoid leaking information about any specific record in the database. Such works have a different goal and model and can be added as a front end in the proposed protocol to provide privacy-preserving answers to the health organization's queries.

6.2 Complexity Analysis

Let N denotes the number of patients in a PHR system. These patients are organized into L groups and each group contains $n = \frac{N}{L}$ patients. Recall that the execution time of range queries, exact matching queries and partial matching queries on a Kd-tree with $\sqrt{N}$ records, will be $O(N^{0.5-1/2d} + m)$, $O(\log N)$ and $O(N^{0.5-s/2d} + m)$, respectively [23] where d is the number of the attributes in the table, s is the number of attributes in the query predicate and m denotes the number of records, reported as the query result. The communication and the computation cost of the protocol is summarized in Table 2 where $T(N)$ indicates the execution time of different types of queries (e.g., exact-matching, partial matching and range matching). The most communication- and computation-intensive operation on the assisting servers is the tree traversal.

6.3 Performance Evaluation

To evaluate the performance of the proposed protocol, we implement a prototype relying on some existing open source projects [24] in Java 1.6. The secret keys p and q of the GM cryptosystem are 256-bit long. Moreover, we employ the publicly available *Breast Cancer* dataset [25]. It has 286 records with 9 categorical attributes. The patient's and the server's side experiments are conducted on an Intel core i5 2.3GHz notebook with 4GB of RAM. The number of patients in each group is fixed at $L = 286$ leading to approximately $81,800$ patients in the PHR system. The shares of the secret key are stored on FPGAs. Decrypting a ciphertext by the cloud server is performed by sending a ciphertext to the

FPGAs. Since the communication is intra-site, we ignore communication delays in the performance evaluation.

To understand the source of the overhead, we measure the query execution time for different types of aggregate SQL queries, but running with only one core enabled. The result is presented in Table 3 and Fig. 4. When k is small, the query time is dominated by Fischlin's protocol, which is independent from the threshold k. Therefore, there is a small difference in the query time when $k = 36$ and $k = 71$. However, as k increases the effect of the threshold decryption becomes more visible and the execution time starts to grow. According to a similar analysis, for small values of k there is a small change in the execution time but as k increases the query time becomes linear with k. Finally, we calculate the execution time of an arbitrary SQL query $k = \frac{n}{4} = 71$. In addition, the execution time of a query heavily depends on the number of comparisons that are performed to traverse the Kd-tree. Therefore, we consider three different scenarios: (1) the worst case scenario is when evaluating predicates targeting a single attribute for interval matching such that all the tree nodes are traversed, (2) the best case scenario is when evaluating predicates targeting all attributes for exact matching, and (3) the real-world scenario is when evaluating predicates targeting multiple attributes for range and exact matching predicates. In the worst case, the time required to evaluate the predicate is 110 seconds (approximately 2 minutes) for each group, whereas in the best case it is 0.3 seconds. In the real-world scenario, we derive the execution time of a SQL query that contains interval matching and exact matching predicates. For each type of predicate, we execute four SQL queries with different number of attributes. The results are presented in Fig. 5. The results indicate that as the number of attributes in the query increases, the execution time decreases due to the smaller search space and the reduction of the number of comparisons. Our experimental results indicate that the proposed protocol would work well with medium size databases (with a total number of 100,000-400,000 patients) and the queries that contain multiple attributes. These results are from a straightforward implementation of the proposed protocol. Further optimizations may lead to a better performance. It is worth to mention that health organizations do not frequently conduct statistical studies on the medical databases (every month or when there is a pandemic). Therefore, the performance of the protocol is acceptable for this type of applications whose goal is to perform search while the absolute privacy of the patients and the health organization is preserved.

Query	**Query Time**(ms)
Select by =	41.93
Select range	216.14
Select sum	33.02
Select max/min	217.32

Table 3: Assisting server latency for different types of SQL queries ($k = \frac{n}{4} = 71$).

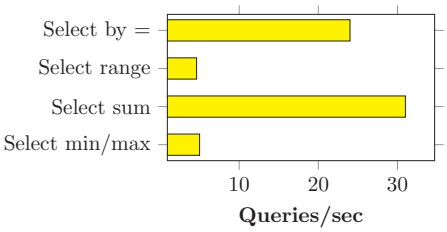


Fig. 4: Query Time

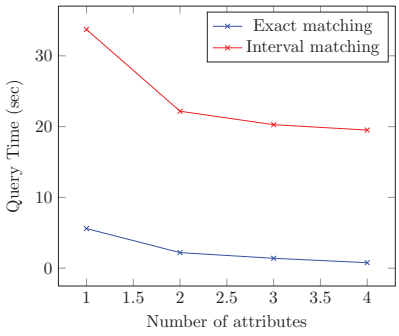


Fig. 5: SQL Query Time (Exact Matching vs Interval Matching)

7 Related Work

Private database outsourcing deals with the problem of hiding database records from an untrusted service provider. To ensure the security and privacy of the data stored in the cloud, most existing approaches rely on encryption [26]. However, using cryptography as a means to protect privacy causes new problems, such as querying the encrypted data. Previous work in querying encrypted data is divided into two categories, one that assumes the existence of a trusted server [27], and another that assumes that the server is semi-trusted [28, 29].Hacigumus *et al.* [28] suggested the addition of secure indexes in each tuple. Although these techniques have been proposed to secure databases hosted in the cloud, they cannot be adopted for this problem for several reasons. First, to evaluate the query on the encrypted data, the health organization must encrypt the query by the same scheme and the same key that are used by the patients and send it to the cloud server. It then forwards the encrypted query to the patients, where the query can be decrypted by the encryption key. Second, a common approach in the existing research proposals is to send a set of encrypted records to the client for filtration and further processing [28, 30]. Therefore, the cloud server may reveal extra information beyond the query result to the client.
Thus, the proposed techniques for secure database outsourcing will not protect the query privacy and the database privacy. Recently, CryptDB [31] has been proposed to execute SQL queries over encrypted data. It depends on a fully trusted component that maintains all the secret and public keys and transforms the users' SQL queries to ones that can be executed over encrypted records. CryptDB has low overhead on query execution time; however, it requires a fully trusted component which is the single point of attack.

8 Conclusion

In this paper, we have presented a protocol that allows executing various types of SQL queries on PHRs stored in the cloud while preserving the privacy of the

patients and the health organization as well. The health records are encrypted using probabilistic encryption schemes, which are semantically secure. The protocol supports aggregate, exact matching and range matching queries. It is based on Fischlin's protocol for private comparison and on two threshold cryptosystems. The implementation result has indicated that the protocol works well with medium size databases and the queries that contain multiple attributes. We have shown that we can execute queries over encrypted data using probabilistic cryptosystems. This opens the door for more research for efficiency in this domain.

References

1. H. Löhr, A. Sadeghi, and M. Winandy. Securing the e-health cloud. In *Proceedings of the International Health Informatics Symposium*, IHI '10, pages 220–229. ACM, 2010.
2. Microsoft health vault, Accessed March, 2013. http://www.healthvault.com/Personal/index.html.
3. Dossia personal health platform, Accessed March, 2013. http://www.dossia.org/.
4. Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of The 14th Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, 1982.
5. M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Proceedings of the Conference on Topics in Cryptology*, pages 457–472, 2001.
6. P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99*, volume 1592, pages 223–238. 1999.
7. Wei Jiang and Chris Clifton. A Secure Distributed Framework for Achieving k-Anonymity. *The VLDB Journal*, 15(4):316–333, November 2006.
8. F. Olumofin and I. Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of the International Conference on Privacy Enhancing Technologies*, PETS'10, pages 75–92. Springer-Verlag, 2010.
9. Samira Barouti, Dima Alhadidi, and Mourad Debbabi. Symmetrically-Private Database Search in Cloud Computing. *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science*, 1:671–678, December 2013.
10. J. Katz and M. Yung. Threshold Cryptosystems Based on Factoring. In *Proceedings of the Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, pages 192–205, 2002.
11. Adam Barnett and Nigel P. Smart. Mental Poker Revisited. In *Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 370–383. Springer Berlin Heidelberg, 2003.
12. Dan Boneh and Matthew Franklin. Efficient Generation of Shared RSA Keys. *J. ACM*, 48(4):702–722, July 2001.
13. Andrew C. Yao. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE Computer Society, 1982.
14. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commumications of the ACM*, 18(9):509–517, September 1975.

15. Takashi Nishide and Kouichi Sakurai. Distributed Paillier Cryptosystem Without Trusted Dealer. In *Proceedings of the 11th International Conference on Information Security Applications*, WISA'10, pages 44–60. Springer-Verlag, 2011.
16. Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust Efficient Distributed RSA-Key Generation. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, STOC '98, pages 663–672. ACM, 1998.
17. K. Eguro and R. Venkatesan. FPGAs for Trusted Cloud Computing. In *FPL*, pages 63–70, 2012.
18. O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, 2001.
19. Yehuda Lindell and Benny Pinkas. Privacy Preserving Data Mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 36–54. Springer-Verlag, 2000.
20. Cynthia Dwork. Differential Privacy: A Survey of Results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, TAMC'08, pages 1–19. Springer-Verlag, 2008.
21. Frank D. McSherry. Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis. In *Proceedings of the 2009 SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.
22. Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and Privacy for MapReduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 297–312. USENIX Association, 2010.
23. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 2008.
24. Martin Joakim Bittel Geisler. *Cryptographic Protocols:: Theory and Implementation*. PhD thesis, Aarhus UniversitetAarhus University,[Enhedsstruktur før 1.7. 2011] Aarhus University, Det Naturvidenskabelige FakultetFaculty of Science, Datalogisk InstitutDepartment of Computer Science, 2010.
25. UCI Machine Learning Repository: Breast Cancer Data Set, April 2012. http://archive.ics.uci.edu/ml/datasets/Breast+Cancer.
26. R. Sion. Secure data outsourcing. In *Proceedings of the conference on Very large data bases*, VLDB '07, pages 1431–1432, 2007.
27. Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik, and Yonghua Wu. A framework for efficient storage security in rdbms. In *Advances in Database Technology - EDBT 2004*, volume 2992 of *Lecture Notes in Computer Science*, pages 147–164. Springer Berlin Heidelberg, 2004.
28. Hakan Hacigümüş, Bala Iyer, and Sharad Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Database Systems for Advanced Applications*, volume 2973 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 2004.
29. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.
30. B. Hore, Sh. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proceedings of the international conference on Very large data bases - Volume 30*, VLDB '04, pages 720–731. VLDB Endowment, 2004.
31. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB : Protecting confidentiality with encrypted query processing. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.