Better Security for Queries on Encrypted Databases

Myungsun Kim[1], Hyung Tae Lee[2], San Ling[2],
Shu Qin Ren[3], Benjamin Hong Meng Tan[2], and Huaxiong Wang[2]

[1] Department of Information Security
The University of Suwon
Hwaseong, South Korea
msunkim@suwon.ac.kr
[2] Division of Mathematical Sciences
School of Physical & Mathematical Sciences
Nanyang Technological University, Singapore
{hyungtaelee,lingsan}@ntu.edu.sg,
TANH0199@e.ntu.edu.sg, hxwang@ntu.edu.sg
[3] Data Center Technologies Division, Data Storage Institute
A*STAR, Singapore
REN_Shuqin@dsi.a-star.edu.sg

Abstract. Private database query (PDQ) processing has received much attention from the fields of both cryptography and databases. While previous approaches to design PDQ protocols exploit several cryptographic tools concurrently, recently the appearance of fully homomorphic encryption (FHE) schemes enables us to design PDQ protocols without the aid of additional tools. However, to the best of our knowledge, all currently existing FHE-based PDQ protocols focus on protecting only constants in query statements, together with the client's data stored in the database server.

In this paper, we provide a FHE-based PDQ protocol achieving better security, protecting query types as well as constants in query statements for conjunctive, disjunctive, and threshold queries with equality comparison. Our contributions are three-fold: First, we present a new security definition that reflects our enhanced security model which additionally protects query types in query statements. Second, we provide a new design for PDQ protocols using FHE schemes. To do this, we come up with a method to homomorphically evaluate our encrypted target queries on the encrypted database. Thereafter, we apply it to construct a protocol and show its security under our enhanced security definition in the semi-honest model. Finally, we provide proof-of-concept implementation results of our PDQ protocol. According to our rudimentary experiments, it takes 40 seconds to perform a query on 2352 elements consisting of 11 attributes of 40-bit using Brakerski-Gentry-Vaikuntanathan's leveled FHE with SIMD techniques for 80-bit security, yielding an amortized rate of just 0.12 seconds per element.

Keywords: Private queries, Encrypted database, Homomorphic encryption

1 Introduction

Cloud computing involves highly durable storage platforms supporting a wide scope of services. One key application is running a relational database in the cloud, e.g., Amazon Relational Database Service [2], but it is not limited to relational databases. Outsourcing databases to a cloud server provides sustainable cost advantages, robustness, and availability. In particular, the need to rapidly deploy leads many enterprises and organizations to move their databases to the cloud, e.g., Akamai Technologies [12]. As a trade-off to these benefits, the main issue that affects outsourcing is maintaining the privacy of information; particularly for those that are sensitive. This makes sense in personal uses of cloud database services as well.

From the perspective of a client which has been storing data in the cloud, two key privacy challenges arise.

– Protection of outsourced data from theft by hackers and workers on the cloud side: Encryption by the client and authenticated access seem to be a straightforward solution. In

particular, homomorphic encryption that supports computing on encrypted data seems to be preferable because the decryption key must not be shared with the server.

– Protection of submitted queries from being disclosed to the server: SQL-like languages may reveal much information about the client's intentions and interest. In other words, learning the client's query details implies learning its possibly sensitive search interest. Moreover, a long history of client queries could allow the server to gradually learn the information in the encrypted database. A naïve solution is to encrypt constants used in the condition clauses in a query statement.

A bit more formally, a private database query (PDQ) is a procedure which on input of an encrypted database $\overline{D}$ with n encrypted tuples $(\overline{\alpha}_1, \ldots, \overline{\alpha}_n)$ and a query statement Q over $\overline{D}$, outputs n' ciphertexts $(\overline{\gamma}_1, \ldots, \overline{\gamma}_{n'})$ while keeping the following two conditions:

1. Without the decryption key, no one can obtain any information about α_i's and γ_i's from any intermediate and final outputs of the procedure, including the encrypted database $\overline{D} = (\overline{\alpha}_1, \ldots, \overline{\alpha}_n)$ and the result set $(\overline{\gamma}_1, \ldots, \overline{\gamma}_{n'})$.
2. Without the decryption key, no one can obtain any information on the conditional clause of Q (e.g., the constant a_1 in an SQL query "`select * from` R `where` $A_1 = a_1$ `and` $A_2 = a_2$" for a schema of degree τ, $\mathsf{R}(A_1, \ldots, A_\tau)$).

The first property can be easily achieved by relying on the security of the exploited encryption scheme (e.g., indistinguishability against chosen plaintext attacks (IND-CPA)). However, the second is relatively not easy to handle from the security's point of view. One main reason is the difficulty of balancing performance and security. For example, in the sense of the relational model, a query Q is partitioned into three parts: the list of needed attributes, the list of relations, and the selection condition. For example, we may represent a query Q as $\boldsymbol{\pi}_{\langle \text{attribute\_list} \rangle}(\boldsymbol{\sigma}_{\langle \text{selection\_condition} \rangle}(\text{relation}))$ using a relational algebra expression. Obviously, all parts of query reveal the client's interest, but taking into account the query costs, i.e., running time, for the specific database management system and database concerned, one may not choose to protect all parts from the server.

For these reasons, we also limit our interest here to protecting the selection condition following existing work on PDQ (e.g., [3, 30]). Thus, the second security property states that the server (and the adversary) should not know information on the selection condition of queries. The most popular approach to provisioning PDQs with the second property is to encrypt the constants in the conditional clause of the query.

However, unfortunately, encrypting the constants only is not sufficient to satisfy the second condition. We note that the above SQL query protects a_1 by encryption, but the attribute names A_1, A_2 and the logical operator '`and`' are still detected by the server. Therefore, the existing works achieve a security model with the second condition altered such that it is sufficient to hide only the constants. The contribution of this paper is to present an efficient technique to protect logical operators as well as constants from the server, which makes it possible to design a new PDQ protocol with better security. We remark that because protecting attributes in the selection condition gives rise to the same performance issue as above, this is beyond our scope. For the same reason, we only focus on conjunctive (and threshold conjunctive) and disjunctive queries with equality comparison.

1.1 System Model

Our system model for PDQ consists of a client and a server: The client who has his/her own database, stores it at the server. When a need arises, he/she sends a query request to the server

and receives a result of the query over the stored database from the server. Here, the client does not want to reveal any partial information about his/her queries and database to the server and/or the adversary.

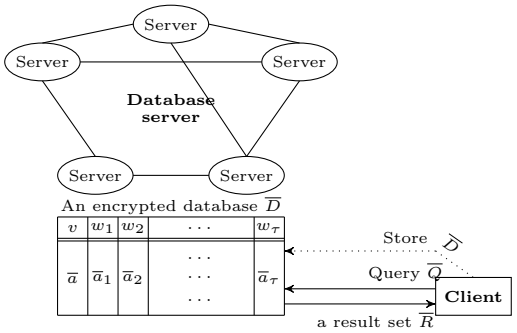

Fig. 1. Our PDQ Model

In this work, we restrict our attention to the case that uses (leveled) fully homomorphic encryption (FHE) since FHE-based query protocols are capable of privately performing search and aggregate queries without the aid of other cryptographic tools.[1] To provide a solution of our system model, we consider the following scenario, shown in Fig. 1 (Throughout the paper, $\overline{m}$ denotes an encryption of m for any message m.): In the initial deployment phase, the client encrypts its database and stores it in the cloud database server. When he/she sends a query request to the server, he/she sends encrypted constants in the query statement Q. Then, the server evaluates the encrypted query with the encrypted database homomorphically and returns an encrypted result set. Here, if the exploited encryption scheme is secure, then no information about the plaintexts in the encrypted database will be revealed, thereby solving the first challenge as in the previous literatures [11, 26].

1.2 Basic Idea

In this paper, we focus on addressing the second challenge in PDQ for conjunctive, disjunctive, and threshold queries with equality comparison. To this end, we come up with a method to homomorphically evaluate our encrypted target queries in the same way, regardless of query type. Our basic idea is as follows: Once the client poses a query statement Q, he/she parses it into the textual part and conditional part.

1. We first observe that the conditional part for conjunction, disjunction, and threshold queries with equality comparison can be represented with the same circuit structure by using an equality test circuit. Let EQTest be a circuit that on input ciphertexts $\overline{m}_1$ and $\overline{m}_2$, outputs $\overline{1}$ if $m_1 = m_2$ and otherwise outputs $\overline{0}$. Then, for $b, c, d, A_i, a_i \in \mathbb{F}_{2^\ell}$ with a finite field $\mathbb{F}_{2^\ell}$ of characteristic 2, we can consider a circuit

$$\overline{d} + \prod_{i=1}^{\tau} [\overline{b} + \mathsf{EQTest}(\overline{A}_i, \overline{a}_i) \cdot \overline{c}] \tag{1}$$

[1] For reasons of efficiency, a leveled FHE scheme that supports computation of arbitrary functions of predetermined polynomial depth may be applied, as in our implementation, instead of pure fully homomorphic encryption (see Section 5).

to represent conjunctive, disjunctive, and threshold queries with equality comparison between A_i and a_i for each $1 \leq i \leq \tau$. Then, the results of the circuit in Equation (1) are determined as in Table 1 with respect to queries and the choice of constants b, c, and d.[2]

Table 1. Results of Eq. (1) for Queries and Constants

Query Type	b	c	d	Result of Eq. (1)
Conjunction	0	1	0	0/1
Disjunction	1	1	1	0/1
Threshold	1	$1+t$	0	t^κ

t: a proper element in a multiplicative group $\mathbb{F}_{2^\ell}^*$

κ: the number of conditions that an element satisfies in the threshold query

2. In Table 1, the results for threshold queries are still $\overline{t^i}$ for a proper element t in a multiplicative group $\mathbb{F}_{2^\ell}^*$ and some non-negative integer i, whereas those for conjunctive and disjunctive queries are $\overline{0}$ and $\overline{1}$. Hence, we need an additional step, evaluating a polynomial, to obtain correct results ($\overline{0}$ or $\overline{1}$) for threshold queries. To make threshold and non-threshold queries indistinguishable, the client forces the server to evaluate an encrypted polynomial $\overline{g}$ for all queries, where

$$g(x) = \begin{cases} x & \text{for non-threshold queries} \\ h(x) & \text{for threshold queries} \end{cases}$$

such that $h(t^\kappa) = 1$ if $T < \kappa \leq \tau$ and 0 if $0 \leq \kappa \leq \tau$ for a threshold T and number of attributes in the database τ.

After the above steps, the server has the intermediate results of the query i.e., $\overline{1}$ for true and $\overline{0}$ for false. Therefore, he/she can obtain the final result by multiplying the encrypted data (the values in the v column in Fig. 1) and the intermediate results.

Therefore, if the client encrypts all constant terms in Step 1 and coefficients of the polynomial g in Step 2, sends all to the server, and lets the server evaluate correctly, then the client will receive an encrypted set of correct results. Informally speaking, in the above, the client can hide his query by relying on the security of the exploited FHE scheme. Our description here is given in the abstract level, but we defer the full description to Section 4.

1.3 Our Results

This paper aims at making the following three contributions.

New security definitions. We provide a new security definition for PDQ that achieves better security than the existing works by additionally considering the protection of logical operators as well as constants in query statements. To this end, we first review existing security definitions for PDQs and describe some of their limitations. Specifically, we recall that existing definitions do not guarantee the privacy of the client's queries (and is therefore not an adequate notion of security for constructing PDQ protocols) and then highlight technical issues. Thereafter, we address this issue by proposing a new definition that provides better security for both type of queries and constants in the conditional part of a query.

[2] See Section 4 for the detail of the element $t \in \mathbb{F}_{2^\ell}^*$.

New constructions. We present a construction for PDQs, that is secure under our new security definition in the semi-honest model. Our protocol can efficiently handle conjunctive queries, disjunctive queries, and threshold conjunctive queries with equality comparison. As mentioned before, our construction mainly relies on the equality test circuit, but pays the price of increasing computation and communication size to achieve a higher level of security.

In fact, our protocol achieves searches in two communication rounds, requires an amount of work from the server that is linear in the number of tuples of databases, and requires a total multiplicative depth of $\lceil \log \ell \rceil + 2\lceil \log(1+\tau) \rceil + 2$ where ℓ is the extension degree of plaintext space of an exploited FHE scheme and τ is the number of attributes of elements in the database. It also requires the total communication cost $n + 4\tau + 1$ from the client where n is the number of tuples in the database.

Experimental study. We provide implementation results of our design of a PDQ protocol and discuss the details in Section 5. As an FHE scheme, we exploit the Brakerski-Gentry-Vaikuntanathan (BGV) leveled FHE scheme [8]. Because our protocol is computationally intensive, we apply some additional techniques to improve the performance of the protocol. For example, we utilize depth-free Frobenius map evaluation for performing polynomial evaluation as well as computing an equality test algorithm on encrypted data. We also employ dynamic programming techniques to boost the computational efficiency in polynomial evaluations. (See Section 5.1 for details.)

We report on our experimental results for various settings in Section 5.3. Further, we provide a comparison of ours with the previous standard query over encrypted data using FHE in [26], that achieves weaker security than ours. From our experiments, for 80-bit security, it requires 40 seconds to perform a query on 2352 elements consisting of 11 attributes of 40-bit, yielding an amortized rate of just 0.12 seconds per element.

1.4 Related Work

A popular PDQ system CryptDB [35] and its offshoot Monomi [39] are two systems that have combined encryption schemes with different properties and a proxy server to translate standard database queries into instructions for a database server to work on stored encrypted data. However, some details such as the type of query received are leaked to the database server based on instructions from the proxy.

Since the appearance of plausible somewhat homomorphic encryption (SHE) and FHE, there were several proposed PDQ solutions [5, 11, 26] based on SHE and FHE. Boneh et al. [5] provided PDQ protocols for conjunctive and threshold conjunctive queries. In their suggestion, they represent each attribute as a polynomial whose roots are the indices of recodes that satisfy the attribute. The coefficients of those polynomials are encrypted and they are manipulated based on the incoming query. Later, Cheon et al. [11] proposed a PDQ protocol for searching on encrypted database using an equality test algorithm on encrypted data. The basic idea of their suggestion is to find a predicate to efficiently represent a search condition and then to evaluate at each FHE ciphertext by applying an equality test algorithm. We note that their work restricts the plaintext space of the exploited FHE to a set $\{0, 1\}$.

Very recently, Kim et al. [26] proposed three PDQ protocols for conjunctive, disjunctive, and threshold conjunctive queries with equality comparison, respectively, and provided the efficiency analysis of their constructions based on the required multiplicative depth to perform equality test between encrypted data using FHE with respect to plaintext spaces of the exploited FHE schemes. They also suggested a communication-efficient method that reduces a communication cost on the server side by adapting the technique to represent a set by a polynomial as in

previous works on private set operations and PDQs [5, 17, 27]. However, the works mentioned above are not also concerned with hiding the type of query that will be evaluated.

For hiding the type of query, Goldwasser et al. [22] proposed multi-input functional encryption (MI-FE) and described a method that allows a user to compute database queries without revealing them to anyone else. Boneh et al. [6] constructed a secret-key MI-FE that is more efficient but they are not practical yet.

A more practical approach to hiding queries is [33] which uses Bloom filters (BF) to make searching efficient and a proxy server that transforms a user's encrypted query into BF indices that are used to extract the data identifiers from the actual database. The database sends the identifiers to the proxy who encrypts and returns the result to the user. However, there is the possibility of false positives due to BF and it leaks some patterns in the submitted queries and results.

In [16, 32], there have been proposed improvements of this result in a scheme called Blind Seer. They add more primitives such as Yao's garbled circuits and oblivious transfer. Although it can support arbitrary Boolean formulas over large databases, the drawback remains that false positives are still possible and it requires a non-constant number of rounds of communication when evaluating the Yao's garbled circuit that is the search tree and in the process leaks the traversal pattern of the tree.

1.5 The Outline of the Paper

Our paper is structured in the following way. In Section 2, we take a look at concepts of FHE and recent results on the required depth for an equality test algorithm using FHE, which are two main components of our proposed protocol. We provide a new security definition of our PDQ model in Section 3. Our new PDQ protocol is presented in Section 4, followed by subsections that show an in-depth performance evaluation and analyze the security. In Section 5, we provide our proof-of-concept implementation and its experimental results with various parameters.

2 Preliminaries

In this section, we briefly introduce concepts of FHE and look into the recent result on the required multiplicative depth to perform an equality test algorithm between encrypted data using FHE, which is a main component of our proposed protocol.

2.1 Notation

Throughout the paper, $a \xleftarrow{\$} A$ denotes that an element a is chosen uniformly and randomly from a set A. For an algorithm A, $A \to a$ denotes that the algorithm A outputs a. Let $\overline{a}$ denote an encryption of a plaintext message a. The set of integers from 1 to a is denoted by $[a]$. We denote by $\lceil a \rceil$ (resp. $\lfloor a \rfloor$) the smallest (resp. largest) integer that is larger (resp. smaller) than or equal to a real number a.

We denote by λ the security parameter and for simplicity, assume that all algorithms take it as input. A function $\nu : \mathbb{N} \to \mathbb{R}$ is negligible in λ if for all positive polynomials $p(\cdot)$ and sufficiently large λ, $\nu(\lambda) \leq \frac{1}{p(\lambda)}$. We use $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ to represent unspecified polynomials and negligible functions in λ, respectively. A probabilistic polynomial-time (PPT) algorithm is a randomized algorithm that runs in time $\mathsf{poly}(\lambda)$.

Database. Let $D = (\alpha_1, \ldots, \alpha_n)$ be a database of n tuples, and each tuple α_i is an ordered list of $\tau+1$ attributes $(v, w_1, \ldots, w_\tau)$. We use $|D|$ to indicate the size of the database D, i.e., $n = |D|$. We then refer to each element by $\alpha_i.v$ or $\alpha_i.w_j$ for $j \in [\tau]$. Here, each attribute $\alpha_i.w_j$ represents

a keyword attribute (e.g., student ID) which is checked against the conditions in a search (e.g., a `where`-clause in SQL). In contrast, the element $\alpha_i.v$ represents a value attribute (e.g., his grade) which is retrieved if the tuple α_i satisfies the search condition. Because we are interested in the privacy of search conditions, we simply assume that the schema has one value attribute (and note that it is straightforward to expand it to multiple value attributes).

Furthermore, we use $\mathsf{id}(\alpha_i)$ to denote the identifier of tuple α_i, which is any string that uniquely identifies it, such as a memory address. We denote by $\mathbf{Id}_D(Q)$ the list consisting of the identifiers of all tuples in D such that a query Q over D is evaluated to true. Let $\boldsymbol{Q}_D[\eta_1, \eta_2]$ be the set of all queries over a database D which have not more than η_1 logical operators (e.g., `and`) and η_2 constants (or literals) at the search condition where $\eta_1 = \mathsf{poly}(\lambda), \eta_2 = \mathsf{poly}(\lambda)$.[3] Lastly, we denote by $\boldsymbol{R}_D(Q)$ a result set from executing a query Q over a database D.

2.2 Fully Homomorphic Encryption

A fully homomorphic encryption (FHE) scheme consists of the following four PPT algorithms, $\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$, and Eval:

- $\mathsf{KeyGen}(\lambda) \rightarrow (pk, ek, sk)$: It takes a security parameter λ as input and outputs a public key pk, an evaluation key ek, and a secret key sk. We assume that pk, sk, and ek each include the information of both the plaintext space $\mathcal{P}$ and ciphertext space $\mathcal{C}$.
- $\mathsf{Enc}(pk, m) \rightarrow \overline{m}$: Given the public key pk and a plaintext message $m \in \mathcal{P}$, it outputs a ciphertext $\overline{m}$.
- $\mathsf{Dec}(sk, \overline{m}) \rightarrow m^* / \perp$: Given the secret key sk and a ciphertext $\overline{m}$, it outputs a message $m^* \in \mathcal{P}$ or $\perp$.
- $\mathsf{Eval}(ek, \varphi, \overline{m}_1, \cdots, \overline{m}_n) \rightarrow \overline{m}_\varphi$: It takes the evaluation key ek, a function $\varphi : \mathcal{P}^n \rightarrow \mathcal{P}$, and a set of n ciphertexts $\overline{m}_1, \cdots, \overline{m}_n$ as inputs and outputs a cihpertext $\overline{m}_\varphi$.

Here, arbitrary functions φ are allowed to be evaluated in the Eval algorithm of pure FHE schemes.

An FHE scheme is said to be IND-CPA secure if it achieves indistinguishability against chosen plaintext adversaries. We use a widely known formulation of IND-CPA security, defined as follows.

Definition 1 (IND-CPA Security) *An FHE scheme is IND-CPA secure if for any polynomial-time adversary $\mathcal{A}$, it holds that*

$$\Big| \Pr[\mathcal{A}(pk, ek, \mathsf{Enc}(pk, m_0)) = 1] - \Pr[\mathcal{A}(pk, ek, \mathsf{Enc}(pk, m_1)) = 1] \Big| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, ek, sk)$ and $m_0, m_1 \in \mathcal{P}$ are chosen by the adversary $\mathcal{A}$.

Since Rivest et al. [36] proposed the concept of FHE in 1978, it remained an open problem to realize a plausible construction for 30 years. In 2009, Gentry proposed the first FHE scheme from ideal lattices [18]. Following his design philosophy, various studies [13,15,40] have been presented on constructing efficient FHE schemes, however they have fairly poor performance. As a solution of efficient FHE, Brakerski and Vaikuntanathan [9] introduced the concept of leveled FHE schemes, which allows the evaluation of functions of at most a pre-determined multiplicative

[3] For example, an SQL statement `select` $*$ `from` ACCOUNT `where` Age $= 25$ `and` Gender $=$ 'male' is in $\boldsymbol{Q}_D[1, 2]$, where Age and Gender are the attribute names of relation ACCOUNT.

depth, instead of arbitrary functions. Shortly after, Brakerski, Gentry, and Vaikuntanathan [8] proposed a leveled FHE scheme over polynomial rings, which has significantly improved performance over the previous schemes.

When an FHE scheme is fixed, the efficiency of evaluating a function is primarily determined by the required levels to evaluate a circuit corresponding to the function. Furthermore, when plaintext spaces of FHE schemes are extension fields, we can further reduce the required multiplicative depth by using FHE schemes that support depth-free automorphism evaluation. We will exploit such FHE schemes [7, 8] in our proposed protocol.

2.3 Equality Test Algorithm

Our protocol will utilize an equality test algorithm on encrypted data using FHE schemes as a building block. An equality test algorithm between two encrypted data, denoted by $\mathsf{EQTest}(\cdot, \cdot)$, is defined as follows: For two messages $m_1, m_2 \in \mathcal{P}$,

$$\mathsf{EQTest}(\overline{m}_1, \overline{m}_2) = \begin{cases} \overline{1} & \text{if } m_1 = m_2 \\ \overline{0} & \text{otherwise.} \end{cases}$$

When an FHE scheme is fixed, the computational efficiency of the equality test algorithm is determined by the required multiplicative depth when evaluated using that FHE scheme. Some works in the literature [10, 11, 26] measured the required multiplicative depth of performing an equality test algorithm on encrypted data using FHE schemes. According to a very recent analysis by Kim et al. [26], it consumes an optimal multiplicative depth $\lceil \log \ell \rceil$ to perform an equality test algorithm between two ciphertexts of ℓ-bit messages, when the plaintext space of the exploited FHE scheme is a field of characteristic 2 and no multiplicative depth is consumed to evaluate the Frobenius map. Therefore, our protocol will utilize FHE schemes, such as [7, 8], whose plaintext space is $\mathbb{F}_{2^\ell}$ and which consumes no multiplicative depth to evaluate the Frobenius map.

3 Our Security Model

In this section, we provide the security model for our PDQ model. We first look at security definitions of previous PDQ protocols and their limitations. Then, we define our new security model for PDQ protocols to address those limitations.

3.1 Revisiting Query Privacy Definitions

While security for database queries is typically captured as the requirement that nothing be leaked beyond the *result of a query* or the access pattern (i.e., the unique identifiers of records that satisfy its query condition), we are not aware of any previous work other than that of [19, 21, 28, 31, 38] that satisfies this intuitive definition. This is simply because with the exception of oblivious RAMs, all the constructions in the literature also reveal whether different searches share the same conditions or not. We refer to this as the *condition pattern* and note that it is clearly revealed by the schemes presented in [11, 30, 35]. Therefore, a more accurate characterization of the security notion achieved for PDQs is that nothing is leaked beyond the *access pattern* and the *condition pattern* (see below for precise definitions).

Limitations of Previous Definitions. To date, one definition of security have been used for PDQ which was introduced by Kantarcioğlu and Clifton [25]: For two messages, M_{Q_1} and M_{Q_2}, associated with respective queries, Q_1 and Q_2, of same size over a database D, any adversary

cannot distinguish between an encryption of M_{Q_1} and M_{Q_2}. Borrowing from cryptographic terminology, we can rephrase it as follows: For all PPT distinguishing algorithms $\mathcal{D}$,

$$\left| \Pr[\mathcal{D}(M_{Q_1}) = 1] - \Pr[\mathcal{D}(M_{Q_2}) = 1] \right| \leq \mathsf{negl}(\lambda). \tag{2}$$

To our knowledge, existing works address the security problem by limiting the security goal to hide private constants of a query. As Olumofin and Goldberg remark (p. 77 of [30]), the literature considers that the textual statement of an SQL query is not private, only the constants that the client provides at the execution time are sensitive and so should be kept secret. However, we believe that protecting only the constants is not enough security for PDQ, since logical operators appearing in a query condition can reveal some information that should not be known to any adversary. Though one might be tempted to remedy the situation by introducing functional encryption (e.g., [22]), it cannot be done in an efficient manner yet.

In the same setting, Hore et al.'s scheme [24] splits a query Q into a public part and a private part, and requires that the private section be executed in a trusted area on the server side. The authors, however, did not further refine it. Regarding existing definitions, Yang et al. [41] consider only very simple queries in $\boldsymbol{Q}_D[0, 1]$. For queries in $\boldsymbol{Q}_D[\eta_1, \eta_2]$, subsequent works including [3–5, 16, 29, 30, 32, 34] in the privacy-preserving query literature support only hiding the private constants of a SQL query (see [29, Section 8]) where $\eta_1, \eta_2 \in \mathbb{N}$. As a consequence, their security definitions are given in tune with their own cases.

In summary, all existing definitions assume that a message M_Q associated with a query Q consists of only the private constants in Q's search condition.

3.2 Our Definitions

We now attempt to address the above issues. Before providing our definition for PDQ, we would like to introduce some supporting notions which we will use.

Semi-Honest Model. In this paper, we restrict an adversary model to the semi-honest case. In this model, we assume that all players faithfully follow the described actions of the protocol and they may try to attain additional information other than the result of the protocol from transcripts produced during the execution of the protocol. (See Chapter 7 of [20] for the details.)

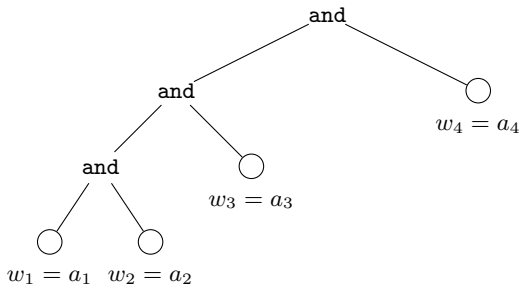

Fig. 2. Example of a Query Tree

Query and Tree Representation. We focus on the conditional content of a query because we exclude the other part of the query (e.g., table name and result attributes) from our targets to be protected in this paper. For this reason, we identify a query with its condition and represent the query into a sequential binary tree representation induced by the condition. For convenience, although a slight abuse of notation, we call it a query tree (e.g., see Fig. 2). Denoting a query

tree for a query as T_Q, for a query $Q \in \boldsymbol{Q}_D[\eta_1, \eta_2]$, we see that η_1 is the number of internal nodes in T_Q and η_2 is the number of leaf nodes in T_Q. Note that we do not consider comparisons between attributes. We denote a vector of internal nodes by $\boldsymbol{o} = (o_1, \ldots, o_{\eta_1})$ and a vector of leaf nodes $\boldsymbol{c} = (c_1, \ldots, c_{\eta_2})$. For simplicity, we assume that each query has the unique sequential tree representation by keeping the order of attributes in the query statement.

Definitions. The interaction between the client and the server is determined by a database and a query with Boolean conditions that the client wishes to search for and that we want to hide from the server. We use a history to indicate a specific execution of such an interaction, following the notion introduced in [14].

Definition 2 (History) *Let $\eta_1, \eta_2 \in \mathbb{N}$. Let D be a database and $Q = (\boldsymbol{o}, \boldsymbol{c})$ be a query in $\boldsymbol{Q}_D[\eta_1, \eta_2]$. A* history *over D and Q is a tuple $H = (D, \boldsymbol{o}, \boldsymbol{c})$ that consists of a database D and $(\boldsymbol{o}, \boldsymbol{c})$ induced by the query tree T_Q.*

Definition 3 (Access Pattern) *Let $\eta_1, \eta_2 \in \mathbb{N}$. Let D be a database and $Q = (\boldsymbol{o}, \boldsymbol{c})$ be a query in $\boldsymbol{Q}_D[\eta_1, \eta_2]$. The* access pattern *induced by a query history $H = (D, \boldsymbol{o}, \boldsymbol{c})$ is the list $\boldsymbol{Id}_D(Q)$ whose query condition is determined by $(\boldsymbol{o}, \boldsymbol{c})$ and attributes of D.*

Definition 4 (Condition Pattern) *Let $\eta_1, \eta_2 \in \mathbb{N}$. Let D be a database and $Q = (\boldsymbol{o}, \boldsymbol{c})$ be a query in $\boldsymbol{Q}_D[\eta_1, \eta_2]$. Then the* condition pattern *induced by a query history $H = (D, \boldsymbol{o}, \boldsymbol{c})$ is a binary tree $T_Q(H)$ such that the structure of T_Q is determined by $\boldsymbol{o}$ and $\boldsymbol{c}$.*

The final notion we study is that of a trace of a history, following the notion from [14]. What we would like to achieve via this notion is to clearly identify which information within a history we are willing to disclose and which information should be hidden from the adversary. Thus, we refer to the former information, that we are willing to reveal, as a trace. The primary difference from existing works is that in our security requirement, the trace does not include the condition pattern. Of course, our definition as well as existing definitions require the trace to carry the access pattern. In addition, the server can see the size of D and the textual content of given queries. Therefore we add these to the trace.

Definition 5 (Trace) *Let $\eta_1, \eta_2 \in \mathbb{N}$. Let D be a database and $Q = (\boldsymbol{o}, \boldsymbol{c})$ be a query in $\boldsymbol{Q}_D[\eta_1, \eta_2]$. The* trace *induced by a query history $H = (D, \boldsymbol{o}, \boldsymbol{c})$ is a tuple $\mathsf{tr}(H) = (|D|, \boldsymbol{Id}_D(Q))$ that consists of the number of tuples $|D|$ and the access pattern $\boldsymbol{Id}_D(Q)$ induced by H.*

Now we are ready to give our security definition for PDQ. Our definition requires that the view of an adversary (i.e., the textual content and the ciphertexts) generated from its chosen history be simulatable given only the trace.

Let $\overline{D} = (\overline{\alpha}_1, \ldots, \overline{\alpha}_n)$ where $\overline{\alpha}_i$ means $\mathsf{Enc}(\alpha_i)$ by an encryption function Enc. Given an encrypted database $\overline{D}$, we can reuse the above definitions and notations except for the use of $\overline{D}$ instead of D. For example, $\mathbf{Id}_{\overline{D}}(Q)$ means a set of identities of $\overline{\alpha}_i$ such that Q's query condition at $\overline{\alpha}_i \in \overline{D}$ is evaluated to true. We define $\mathcal{V}$ as the view of an adversary $\mathcal{A}$ (e.g., the server) over the transcript of the interactions between the client and the server along with the public knowledge. Specifically, $\mathcal{V} = (\overline{D}, Q, \mathbf{Id}_{\overline{D}}(Q), \boldsymbol{R}_{\overline{D}}(Q))$. Accordingly, the trace over the encrypted database $\overline{D}$ means $\mathsf{tr}(H) = (|\overline{D}|, \mathbf{Id}_{\overline{D}}(Q))$.

Definition 6 (Query Condition Privacy) *Let the client and server engage in a query processing protocol π that computes a functionality $f(\mathsf{in}_c, \mathsf{in}_s) = (\mathsf{out}_c, \mathsf{out}_s)$ where $\mathsf{in}_c, \mathsf{out}_c$ (resp., $\mathsf{in}_s, \mathsf{out}_s$) denote input and output of client (resp., server), respectively. Let $\mathcal{V}_\pi$ denote the view of server during the execution of the protocol π. More precisely, the server's view is formed by*

its input $\mathsf{in}_s = \overline{D}$*, internal random coin tosses* r_s*, messages* Q *passed between client and server during protocol execution:*

$$\mathcal{V}_\pi = (\overline{D}, r_s, Q, \boldsymbol{Id}_{\overline{D}}(Q)).$$

Then, we say that a query processing protocol π *achieves* query-condition privacy *if for all databases* D*, queries* Q *over* D*, and PPT algorithms* $\mathcal{A}$*, there exists a PPT algorithm (i.e., the simulator)* $\mathcal{S}$ *such that for all* $\mathcal{V}_\pi$ *and traces* $\mathsf{tr}(H)$ *over* $\overline{D}$*, and for any function* f*:*

$$\{\mathcal{S}(\mathsf{tr}(H), f(D,Q))\} \stackrel{c}{\approx} \left\{\mathcal{V}_\pi, \mathsf{out}_s = \boldsymbol{R}_{\overline{D}}(Q)\right\}$$

where $\stackrel{c}{\approx}$ *denotes computational indistinguishability.*

By the definition of condition privacy, we require that all information on the database D and submitted query Q over D that can be computed by the adversary (i.e., the server) from the transcript of the interactions with the client can also be computed by using public knowledge (i.e., the trace). Intuitively, the notion captures that if a query is condition-private, then it does not leak any information beyond the information that is allowed to leak to the adversary.

4 Our Protocol for Queries with Better Security

In this section, we present our PDQ protocol for conjunction, disjunction, and threshold queries on encrypted databases using FHE schemes. Subsequently, we look into the correctness, the efficiency, and the security of our proposed protocol.

4.1 Preparation

We assume that an encrypted database $\overline{D} = \overline{\alpha}_1 \| \overline{\alpha}_2 \| \cdots \| \overline{\alpha}_n$ using a special FHE scheme is given, where $\alpha_i = (\alpha_i.v, \alpha_i.w_1, \cdots, \alpha_i.w_\tau)$ with $\alpha_i.v \in \{0,1\}^{\ell-1}$ and $\alpha_i.w_j \in \{0,1\}^\ell$ for all $i \in [n]$, $j \in [\tau]$ and where $\overline{\alpha}_i$ is a component-wise encryption of α_i, i.e., $\overline{\alpha}_i = (\overline{\alpha_i.v}, \overline{\alpha_i.w}_1, \cdots, \overline{\alpha_i.w}_\tau)$. As mentioned before, for convenience, we frequently use v_i and w_{ij} in place of $\alpha_i.v$ and $\alpha_i.w_j$, respectively.

For the correctness of our protocol, we assume that $v_i \neq 0$ for all $i \in [n]$. To handle this issue, we exploit an FHE scheme with the plaintext space $\mathbb{F}_{2^\ell}$, not $\mathbb{F}_{2^{\ell-1}}$, and assume that each v_i is encoded as $1\|v_i$ in advance, where $a\|b$ denotes the concatenation of strings a and b. Unless confusion arises, we omit encoding and decoding steps between v_i and $1\|v_i$ in our protocol.

For the efficiency, we exploit FHE schemes that consume no multiplicative depth to evaluate the Frobenius map and its plaintext space is $\mathbb{F}_{2^\ell}$, such as [7, 8]. In this case, the EQTest algorithm that is a main component of our protocol, consumes $\lceil \log \ell \rceil$ multiplicative depth from the analysis in [26]. Throughout the paper, $p(x)$ denotes an irreducible polynomial of degree ℓ where $\mathbb{F}_{2^\ell}$ is isomorphic to $\mathbb{F}_2[x]/((p(x))$. $t \in \mathbb{F}_{2^\ell}$ denotes a root of $p(x)$ and it is predetermined before beginning the protocol.

To encrypt an ℓ-bit messages $a = (a_\ell, \cdots, a_1) \in \{0,1\}^\ell$ using an FHE scheme with the plaintext space $\mathbb{F}_{2^\ell}$, we first encode the message a to $\sum_{i=0}^{\ell-1} a_{i+1} t^i \in \mathbb{F}_{2^\ell}$. Then, we can write an encryption of the message a as

$$\overline{a} := \mathsf{Enc}\left(pk, \sum_{i=0}^{\ell-1} a_{i+1} t^i\right)$$

where pk is the public key of the exploited FHE scheme.

Henceforth, $\overline{a} + \overline{b}$ and $\overline{a} \cdot \overline{b}$ denote operations between ciphertexts that they preserve addition and multiplication on encrypted data, respectively. We remark that $\overline{1} + \overline{1} = \overline{0}$ because it is assumed that the plaintext space of the exploited FHE scheme in our protocol is $\mathbb{F}_{2^\ell}$.

4.2 Our Protocol

Descrption. Now, we present the description of our PDQ protocol between the client and the server.

1. The client performs as follows:
   (a) Depending on the type of query,
   - Conjunction: For a query to return v_i's such that $\bigwedge_{j\in J}(\alpha_i.w_j = a_j)$,
     i. For each $j \in [\tau]$,
     - If $j \in J$, set $b_j = 0$ and $c_j = 1$.
     - Otherwise, set $b_j = 1$, $c_j = 0$, and $a_j \xleftarrow{\$} \{0,1\}^\ell$.

     ii. Set $d_i = 0$ for all $i \in [n]$.
     iii. Compute a polynomial $g \in \mathbb{F}_{2^\ell}[x]$ such that $g(1) = 1$ and $g(0) = 0$.
   - Disjunction: For a query to return v_i's such that $\bigvee_{j\in J}(\alpha_i.w_j = a_j)$,
     i. For each $j \in [\tau]$,
     - If $j \in J$, set $b_j = 1$ and $c_j = 1$.
     - Otherwise, set $b_j = 1$, $c_j = 0$, and $a_j \xleftarrow{\$} \{0,1\}^\ell$.

     ii. Set $d_i = 1$ for all $i \in [n]$.
     iii. Compute a polynomial $g \in \mathbb{F}_{2^\ell}[x]$ such that $g(1) = 1$ and $g(0) = 0$.
   - Threshold conjunction: For a query to return v_i's such that $\kappa > T$ where T is a non-negative integer and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$.
     i. For each $j \in [\tau]$,
     - If $j \in J$, set $b_j = 1$ and $c_j = t + 1$.
     - Otherwise, set $b_j = 1$, $c_j = 0$, and $a_j \xleftarrow{\$} \{0,1\}^\ell$.

     ii. Set $d_i = 0$ for all $i \in [n]$.
     iii. Compute a polynomial $g \in \mathbb{F}_{2^\ell}[x]$ such that $g(t^\kappa) = \begin{cases} 1 & \text{if } T < \kappa \le \tau \\ 0 & \text{if } 0 \le \kappa \le T. \end{cases}$

   (b) The client encrypts a_j, b_j, c_j, d_i, and g_k for all $i \in [n]$, $j \in [\tau]$, and $k \in (\{0\} \cup [\tau])$ where $g(x) = \sum_{k=0}^{\tau} g_k x^k$ and sends them to the server.
2. The server performs as follows:
   (a) computes

$$\overline{\beta}_{ij} = \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j) \cdot \overline{c}_j \tag{3}$$

   for all $i \in [n]$ and $j \in [\tau]$.
   (b) computes $\overline{\zeta}_i = \overline{d}_i + \prod_{j\in[\tau]} \overline{\beta}_{ij}$ for all $i \in [n]$.
   (c) computes $\overline{\gamma}_i = \overline{g}(\overline{\zeta}_i) \cdot \overline{v}_i$ for all $i \in [n]$.
   (d) sends $\{\overline{\gamma}_1, \cdots, \overline{\gamma}_n\}$ to the client.
3. The client obtains $\gamma_1, \cdots, \gamma_n$ by decrypting $\overline{\gamma}_i$'s using the secret key of FHE.

Correctness. The following theorem shows the correctness of our proposed protocol.

Theorem 1. *Let Q be a query over an encrypted database $\overline{D}$ that the client submits at Step 1 of our proposed protocol, where Q is one of queries for conjunctive, disjunctive, and threshold queries with equality comparison. Then, the client obtains the result set $\boldsymbol{R}_D(Q)$ for the query Q over $\overline{D}$ after the execution of the protocol.*

Proof. 1. Conjunction: At Step 2 (a), for a conjunction query $\bigwedge_{j\in J}(\alpha_i.w_j = a_j)$,
– If $j \in J$,

$$\begin{aligned}\overline{\beta}_{ij} &= \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{c}_j \\ &= \overline{0} + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{1} \\ &= \begin{cases}\overline{1} & \text{if } \alpha_i.w_j = a_j \\ \overline{0} & \text{if } \alpha_i.w_j \neq a_j.\end{cases}\end{aligned}$$

– If $j \neq J$,

$$\begin{aligned}\overline{\beta}_{ij} &= \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{c}_j \\ &= \overline{1} + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{0} = \overline{1}.\end{aligned}$$

Hence, at Step 2 (b), for each $i \in [n]$,

$$\overline{\zeta}_i = \overline{d}_i + \prod_{j\in[\tau]} \overline{\beta}_{ij} = \begin{cases}\overline{1} & \text{if } \alpha_i.w_j = a_j \text{ for all } j \in [\tau] \\ \overline{0} & \text{otherwise}\end{cases}$$

because $d_i = 0$ and

$$\overline{\gamma}_i = \overline{g}(\overline{\zeta}_i)\cdot \overline{v_i} = \begin{cases}\overline{v}_i & \text{if } \alpha_i.w_j = a_j \text{ for all } j \in [\tau] \\ \overline{0} & \text{otherwise}\end{cases}$$

because $g(0) = 0$ and $g(1) = 1$.

2. Disjunction: At Step 2 (a), for a disjunction query $\bigvee_{j\in J}(\alpha_i.w_j = a_j)$,
– If $j \in J$,

$$\begin{aligned}\overline{\beta}_{ij} &= \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{c}_j \\ &= \overline{1} + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{1} \\ &= \begin{cases}\overline{1} & \text{if } \alpha_i.w_j \neq a_j \\ \overline{0} & \text{if } \alpha_i.w_j = a_j.\end{cases}\end{aligned}$$

– If $j \neq J$,

$$\begin{aligned}\overline{\beta}_{ij} &= \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{c}_j \\ &= \overline{1} + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j)\cdot \overline{0} = \overline{1}.\end{aligned}$$

Hence, at Step 2 (b), for each $i \in [n]$,

$$\overline{\zeta}_i = \overline{d}_i + \prod_{j\in[\tau]} \overline{\beta}_{ij} = \begin{cases}\overline{0} & \text{if } \alpha_i.w_j \neq a_j \text{ for all } j \in [\tau] \\ \overline{1} & \text{otherwise}\end{cases}$$

because $d_i = 1$ and

$$\overline{\gamma}_i = \overline{g}(\overline{\zeta}_i)\cdot \overline{v}_i = \begin{cases}\overline{0} & \text{if } \alpha_i.w_j \neq a_j \text{ for all } j \in [\tau] \\ \overline{v}_i & \text{if } \bigvee_{j\in J}(\alpha_i.w_j = a_j)\end{cases}$$

because $g(0) = 0$ and $g(1) = 1$.

3. Threshold Conjunction: At Step 2 (a), for a query to return v_i's such that $\kappa > T$ where T is a non-negative integer and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$,
   – If $j \in J$,

$$\begin{aligned}\overline{\beta}_{ij} &= \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j) \cdot \overline{c}_j \\ &= \overline{1} + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j) \cdot \overline{t+1} \\ &= \begin{cases} \overline{t} & \text{if } \alpha_i.w_j = a_j \\ \overline{1} & \text{if } \alpha_i.w_j \neq a_j. \end{cases}\end{aligned}$$

   – If $j \neq J$,

$$\begin{aligned}\overline{\beta}_{ij} &= \overline{b}_j + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j) \cdot \overline{c}_j \\ &= \overline{1} + \mathsf{EQTest}(\overline{\alpha_i.w}_j, \overline{a}_j) \cdot \overline{0} = \overline{1}.\end{aligned}$$

   Hence, at Step 2 (b), for each $i \in [n]$,

$$\overline{\zeta}_i = \overline{d}_i + \prod_{j \in [\tau]} \overline{\beta}_{ij} = \overline{0} + \prod_{j \in [\tau]} \overline{\beta}_{ij} = \overline{t^{\kappa}}$$

   where $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$ and

$$\overline{\gamma}_i = \overline{g}(\overline{\zeta}_i) \cdot \overline{v_i} = \begin{cases} \overline{v}_i & \text{if } T < \kappa \leq \tau \\ \overline{0} & \text{if } 0 \leq \kappa \leq T \end{cases}$$

   by the definition of the polynomial g.

From the above, we show that the client obtains the correct values once he decrypts $\overline{\gamma}_i$'s for all $i \in [n]$ and therefore the client obtains the correct result set for each query over the database. □

4.3 Efficiency

Now, we analyze the efficiency of our proposed protocol.

Computational Cost. We evaluate the computational cost of our proposed protocol in terms of the required multiplicative depth. At Step 2 (a), it requires $\lceil \log \ell \rceil + 1$ multiplicative depth to compute $\overline{\beta}_{ij}$ because the EQTest algorithm consumes $\lceil \log \ell \rceil$ multiplicative depth when the exploited FHE scheme of plaintext space $\mathbb{F}_{2^\ell}$ consumes no multiplicative depth to evaluate the Frobenius map [26]. At Step 2 (b), it takes $\lceil \log \tau \rceil$ multiplicative depth to compute $\prod_{j \in [\tau]} \overline{\beta}_{ij}$. Computing $\overline{\gamma}_i$ at Step 2 (c) consumes $\lceil \log (1 + \tau) \rceil + 1$ multiplicative depth. [4] Thus, the total required multiplicative depth is approximately $\lceil \log \ell \rceil + 2\lceil \log \tau \rceil + 2$.

Communication Cost. At Step 1 (b), the client sends encryptions of a_j, b_j, c_j, d_i, and g_k for all $j \in [\tau]$, $i \in [n]$, and $k \in (\{0\} \cup [\tau])$ where $g(x) = \sum_{k=0}^{\tau} g_k x^k$. Hence, the communication cost on the client side is $n + 4\tau + 1$ ciphertexts of the exploited FHE scheme. On the other hand, the server sends $\overline{\gamma}_i$'s for all $i \in [n]$ to the client at Step 2 (d). Hence, the communication cost on the server side is n ciphertexts of the exploited FHE scheme.

[4] In fact, we can further reduce the required multiplicative depth for computing Step 2 (c) to $\lceil \log \lceil \log \tau \rceil \rceil + 1$ by applying Frobenius map. (See Section 5.1 for the detail.) Then, the total required multiplicative depth of our protocol is also slightly reduced to $\lceil \log \ell \rceil + \lceil \log \tau \rceil + \lceil \log \lceil \log \tau \rceil \rceil + 2$.

Remark 1. When an upper bound δ on the selectivity of a query over a database of n elements is known, we can reduce the communication cost on the server side to $\lfloor \delta n \rfloor$ by applying Kim et al.'s suggestion [26, Section 5] to represent a result set by a polynomial at the cost of $\lceil \log n \rceil$ additional multiplicative depth. However, to apply their technique with SIMD, we require additional techniques that impose heavy computational costs, e.g., extracting a ciphertext of each message from a packed ciphertext. Hence, we do not consider to apply their technique in this paper.

4.4 Security Analysis

In this section, we analyze the security of our proposed protocols. We show that the construction of our PDQ protocol in Section 4.2 satisfies the security definition in Section 3 (see Definition 6). Recall that in our proposed protocol the client encodes his/her queries differently if the type of query is different, but the server always performs the same computation regardless.

Theorem 2. *Assume that the exploited FHE scheme is IND-CPA secure, then our PDQ protocol achieves the query-condition privacy in Definition 6.*

Proof. It suffices to build a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$ and functions f, given the trace of query $\mathsf{tr}(H)$ over an encrypted database $\overline{D}$ by the exploited FHE scheme, $\mathcal{S}$ can simulate the adversary's view $\mathcal{A}(\mathcal{V})$ with non-negligible probability. We assume that $\overline{D}$ is already in place. Of course, the adversary can observe the generation of the encrypted database $\overline{D}$, but it has no extra information except for the size of $\overline{D}$ (i.e., $n = |\overline{D}|$) since the exploited FHE scheme is IND-CPA secure.

Specifically, we show that $\mathcal{S}$ taking as input $\mathsf{tr}(H)$ can generate a view $\mathcal{V}^*$ which is computationally indistinguishable from $\mathcal{V}$ which, in turn, is the real view of the adversary $\mathcal{A}$. By $\mathcal{V}^*$, we mean

$$\mathcal{V}^* = \left(\overline{D^*} = \{\overline{\alpha_i^*}\}_{i\in[n]}, Q^*, \mathbf{Id}_{\overline{D^*}}(Q^*), \boldsymbol{R}_{\overline{D^*}}(Q^*)\right).$$

Now $\mathcal{S}$ constructs the simulated view $\mathcal{V}^*$ as follows.

- Generating $\overline{D^*}$. $\mathcal{S}$ first initializes $\overline{D^*} \leftarrow \varnothing$. For each $\alpha_{i\in[n]}^* \xleftarrow{\$} \mathcal{P}$, it computes $\overline{\alpha_i^*}$ with the public key pk and a randomness r_i^*, and set $\overline{D^*} \leftarrow \overline{D^*} \cup \overline{\alpha_i^*}$. Indeed each tuple α_i^* consists of $\tau + 1$ attributes, but because their values are clear from the context we omit their full descriptions.
- Generating Q^*. From the query Q in the given view $\mathcal{V}$, $\mathcal{S}$ can determine the same index set $[\tau]$ as in $\mathcal{V}$. Then for each $j \in [\tau]$, it computes $\overline{a_j^*}, \overline{b_j^*}$, and $\overline{c_j^*}$ with $a_j^* \xleftarrow{\$} \mathcal{P}, b_j^* \xleftarrow{\$} \mathcal{P}$, and $c_j^* \xleftarrow{\$} \mathcal{P}$ using FHE encryption under the public key pk in the same way as above. Similarly, it computes $\{\overline{d_i^*}\}_{i\in[n]}$ under the public key pk for $d_i^* \xleftarrow{\$} \mathcal{P}$. Finally, the client chooses a polynomial $g^*(x) \xleftarrow{\$} \mathbb{F}_{2^\ell}[x]$ such that $\deg(g^*) = \tau$ and computes an encryption of its coefficients $\overline{g_k^*}$ for $k \in \{0\} \cup [\tau]$.
- Generating $\mathbf{Id}_{\overline{D^*}}(Q^*)$. $\mathcal{S}$ constructs a set of unique identifiers $\{\mathsf{id}(\alpha_i^*)\}_{i\in[n]}$. It takes the set as $\mathbf{Id}_{\overline{D^*}}(Q^*)$.
- Generating $\boldsymbol{R}_{\overline{D^*}}(Q^*)$. For each $i \in [n]$, $\mathcal{S}$ generates $\gamma_i^* \xleftarrow{\$} \mathcal{P}$, and computes $\overline{\gamma_i^*}$ under the public key pk in the same way as above. It then uses the collection of the encryptions as $\boldsymbol{R}_{\overline{D^*}}(Q^*)$.

We show that $\mathcal{V}^*$ is computationally indistinguishable from $\mathcal{V}$ by comparing them in a component-by-component manner. It is easy to check that if the exploited FHE scheme is IND-CPA secure, then $\overline{D}$ and $\overline{D^*}$ are computationally indistinguishable. Next let us consider an

actual query Q and a simulated query Q^*, where two queries have the same textual content. Then we see that

$$\begin{aligned} \{\overline{a}_j, \overline{b}_j, \overline{c}_j\}_{j\in[\tau]} &\overset{\text{c}}{\approx} \{\overline{a_j^*}, \overline{b_j^*}, \overline{c_j^*}\}_{j\in[\tau]}, \\ \{\overline{d}_i\}_{i\in[n]} &\overset{\text{c}}{\approx} \{\overline{d_i^*}\}_{i\in[n]}, \\ &\text{and} \\ \{\overline{g}_k\}_{0\le k\le\tau} &\overset{\text{c}}{\approx} \{\overline{g_k^*}\}_{0\le k\le\tau} \end{aligned}$$

if the exploited FHE scheme is IND-CPA secure, where $\overset{\text{c}}{\approx}$ means computational indistinguishability. For $\mathbf{Id}_{\overline{D}}(Q)$ and $\mathbf{Id}_{\overline{D^*}}(Q^*)$, it is not hard to see that $\mathsf{id}(\overline{\alpha}_i)$ is indistinguishable from $\mathsf{id}(\overline{\alpha_i^*})$, if id is assumed to be a pseudorandom function. Note that we did not specify any assumption to the function id since hiding the access pattern is beyond our interest. Finally for all $i \in [n]$, the indistinguishability between $\overline{\gamma}_i$ in the result set and $\overline{\gamma_i^*}$ in the simulated result set is straightforward. Thus, we can conclude the proof of Theorem 2. □

5 Implementation

In this section, we provide implementation results of our proposed PDQ protocol for several scenarios.

5.1 Techniques for Speeding Up

In our implementation, we employed the BGV encryption scheme [8] as the underlying FHE scheme and exploited two additional techniques to improve the performance of the protocol.

1. We apply depth-free Frobenius maps when evaluating the polynomial g on a ciphertext as well as in the equality test algorithm. Frobenius maps can be used to compute powers of two of any ciphertext without consuming levels in case of BGV encryption scheme of plaintext space $\mathbb{F}_{2^\ell}$. This in turn means that we can reduce the number of multiplications required to evaluate a polynomial on a ciphertext. For each integer j, $j = \sum_{i=1}^{\lceil \log j \rceil} b_i \cdot 2^i$ and $x^j = \prod_{i=1}^{\lceil \log j \rceil} b_i \cdot x^{2^i}$ with $b_i \in \{0, 1\}$. This means that any monomial x^j can be computed in $\lceil \log \lceil \log j \rceil \rceil$ multiplications when Frobenius maps are used to pre-compute x^{2^i} for all $0 \le i \le \lceil \log j \rceil$. Thus, for $\bar{g}$ of degree τ, $\bar{g}$ is computed in $\lceil \log \lceil \log \tau \rceil \rceil$ levels.
2. Our implementation uses the dynamic programming paradigm to reduce the number of multiplications that polynomial evaluation requires. For monomials $\bar{\zeta}^i$, where the Hamming weight of the binary representation of i is a power of two, we keep them aside before evaluating $\bar{g}_i \cdot \bar{\zeta}$ so that they can be used subsequently when evaluating higher degree monomials. The reason we only keep these monomials specifically is to ensure that each monomial is evaluated using the lowest number of levels.

5.2 Experiment Setting

Test Environment. The testing platform was a server with Intel® Core™ i7-4790 @ 3.60 GHz with 8GB RAM. The implementation of our protocol was written in C++. We realized the BGV leveled FHE scheme using HElib [23], NTL 9.7 [37], and GMP Library 6.1.0 [1].

Dataset Generation. A range of databases were randomly generated using a Python script and each element of the databases utilized in the experiments consists of one value attribute and 11 keyword attributes. The maximum size of the entries are fixed at 40-bit. We apply a SIMD technique, so each column in the database is packed and encrypted into a set of ciphertexts with the protocol running over these ciphertexts.

Parameter Selection for BGV Scheme. We employed the BGV scheme with 17 levels for ours and 15 for Kim et al.'s [26] as the underlying FHE schemes. We set the plaintext space of the utilized BGV scheme to be slightly larger and not exact. This is because the parameters of the BGV scheme are determined by a security level, required depth, plaintext size, and so on, but the scope of parameters that we can select is quite narrow and the plaintext space cannot be exact.

An important parameter for the HElib implementation of the BGV scheme is an integer m, which is used to evaluate the Fast Fourier Transform (FFT) of elements in $\mathcal{P}$ and $\mathcal{C}$. This allows ciphertext operations to be performed more efficiently than the standard coefficient representation. To find a set of suitable parameters for plaintext space $\mathbb{F}_{2^{40}}$, we conducted an exhaustive search on integers m of up to 50000 to obtain ring dimensions $\phi(m)$ for the Euler totient function ϕ, its corresponding plaintext space $\mathbb{F}_{2^\ell}$ and number of plaintext slots available. But, we could not obtain any that was exactly $\mathbb{F}_{2^{40}}$. Instead, we chose several ring dimensions that support plaintext spaces larger than $\mathbb{F}_{2^{40}}$. Strictly speaking, this is not an optimal technique because a larger plaintext space decreases the number of plaintext slots in a ciphertext and so amortized performance gets worse. Despite that, the overall performance might be better since there exists no small ring dimensions that are suitable for plaintext space $\mathbb{F}_{2^{40}}$.

5.3 Experimental Results for Various Settings

Now, we provide implementation results of our PDQ protocols. First, Table 2 shows the results of some experiments with 40-bit plaintexts and a database that fills up one ciphertext. A large majority (about 80-85%) of the time taken by the protocol is on the equality tests on encrypted data. This is not surprising because it is one of the larger circuits in the protocol. Furthermore, every ciphertext has to undergo this portion of the protocol whereas the rest of the protocol processes only one column worth of data and that is only 1/11 of the database. On the client side, it takes about 2-4 seconds to generate the encrypted query for various security levels and 0.02-0.05 seconds to decrypt the results of the protocol.

Table 2. Performance Results of Our PDQ Protocol for Various Security Levels

λ	m	$\phi(m)$	$\mathcal{P}$	# of Slots in a Ciphertext	Query Encrypt (Client)	EQTest (Server)	Total Time (Server)	Amortised Time (Server)	Result Decrypt (Client)
80	14491	14112	$\mathbb{F}_{2^{42}}$	336	2.00 sec	33.66 sec	39.61 sec	0.12 sec	0.02 sec
99	30705	15488	$\mathbb{F}_{2^{44}}$	352	4.00 sec	64.29 sec	75.92 sec	0.22 sec	0.04 sec
104	17173	15840	$\mathbb{F}_{2^{60}}$	264	3.00 sec	64.17 sec	73.85 sec	0.28 sec	0.04 sec
118	31695	16896	$\mathbb{F}_{2^{44}}$	384	4.00 sec	67.96 sec	80.90 sec	0.21 sec	0.05 sec
125	27393	17424	$\mathbb{F}_{2^{44}}$	396	4.00 sec	67.90 sec	80.69 sec	0.20 sec	0.05 sec

Each element consists of 11 attributes of 40-bit entries and we used a leveled BGV scheme of 17 levels.
λ: security parameter (bit), $\mathcal{P}$: the plaintext space of the utilized BGV scheme
m: parameter for FFT, $\phi(m)$: the dimension of the polynomial ring of the utilized BGV scheme

The general trend of Table 2 is shown in Fig. 3. The time taken to run the protocol increases as security level increases, although the increase is slight after 80-bit security. This could be due to the high m that is used to achieve those levels of security; m is 14491 for 80-bit security but greater than 27000 for 99, 118 and 125-bit security. At 104-bit security, there are fewer slots compared to the rest so the amortised time per row actually increased despite taking similar time to the other security levels. Optimal parameters for the system are hard to determine as the parameters of BGV FHE encryption schemes depend on the integer m and its corresponding ring dimension $\phi(m)$ and plaintext requirements.

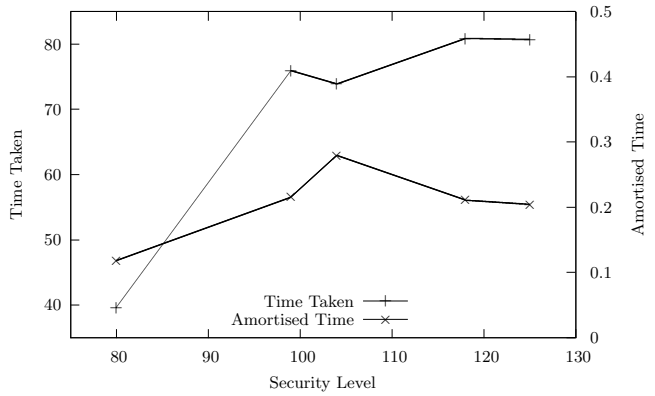

Fig. 3. Timings for Security Levels

Second, Table 3 and Fig. 4 show amortised times of our protocol for various security levels and database sizes. The results show that amortised time of the protocol decreased when the database increased to around 2048. This is likely due to the fact that the first computation usually causes some important reusable data to be moved into memory and then subsequently left there. This means that future computations can save the action of fetching these data again and so take less time to complete.

Table 3. Performance Results for Various Database Sizes

λ	m	$\phi(m)$	$\mathcal{P}$	# of Slots in a Ciphertext	Amortised Time I (# of DB Elements)	Amortised Time II (# of DB Elements)	Amortised Time III (# of DB Elements)
80	14491	14112	$\mathbb{F}_{2^{42}}$	336	0.118 sec (336)	0.097 sec (2352)	0.117 sec (16464)
99	30705	15488	$\mathbb{F}_{2^{44}}$	352	0.216 sec (352)	0.163 sec (2112)	0.214 sec (16544)
104	17173	15840	$\mathbb{F}_{2^{60}}$	264	0.280 sec (264)	0.212 sec (2112)	0.278 sec (16632)
118	31695	16895	$\mathbb{F}_{2^{44}}$	384	0.211 sec (384)	0.158 sec (2304)	0.208 sec (16512)
125	27393	17424	$\mathbb{F}_{2^{44}}$	396	0.204 sec (396)	0.156 sec (2376)	0.201 sec (16632)

Each element consists of 11 attributes of 40-bit entries and we used a leveled BGV scheme of 17 levels.
λ: security parameter (bit), $\mathcal{P}$: the plaintext space of the utilized BGV scheme
m: parameter for FFT, $\phi(m)$: the dimension of a polynomial ring of the utilized BGV scheme

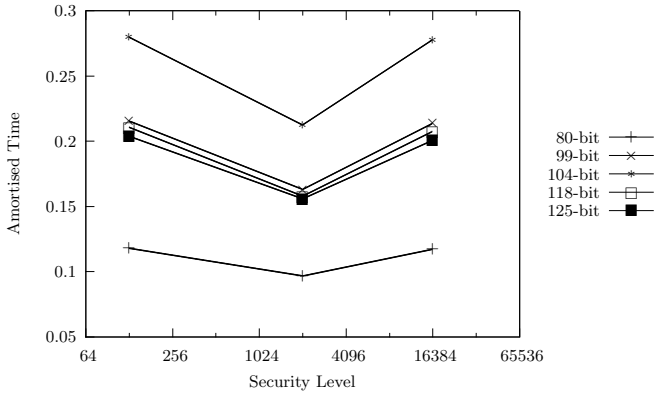

Fig. 4. Timing Results of Our Protocol for Plaintext Sizes

However, this savings is not evident when the database size was pushed to 16384 elements. In this case, the 8 GB RAM was probably insufficient and there is some shuffling of data from the disk drives to the cache and the advantage of reusing data is offset. With more RAM, the savings will be seen again and this shows that small databases can be worked on quite efficiently with consumer level computers.

5.4 Comparison with the Previous Work

The current work improves the security of Kim et al.'s protocol [26] by protecting the user's query type as well as constants in the query statement. This incurs some performance overhead since additional terms have to be encrypted and the number of homomorphic operations between ciphertexts is increased. For more precise comparison, we also implement ours and Kim et al.'s for 3-out-of-11 threshold queries.

Table 4. Timings on the Server Side of Ours and Kim et al.'s for 3-out-of-11 Threshold Query

Our Protocol							Kim et al.'s Protocol						
λ	m	$\phi(m)$	$\mathcal{P}$	# of Slots†	Total Time	Amortized Time	λ	m	$\phi(m)$	$\mathcal{P}$	# of Slots†	Total Time	Amortized Time
80	14491	14112	$\mathbb{F}_{2^{42}}$	336	39.61 sec	0.12 sec	86	15481	12960	$\mathbb{F}_{2^{45}}$	288	31.70 sec	0.11 sec
99	30705	15488	$\mathbb{F}_{2^{44}}$	352	75.92 sec	0.22 sec	93	13367	13366	$\mathbb{F}_{2^{41}}$	326	26.54 sec	0.08 sec
104	17173	15840	$\mathbb{F}_{2^{60}}$	264	73.84 sec	0.28 sec	104	14491	14112	$\mathbb{F}_{2^{42}}$	336	29.89 sec	0.09 sec
118	31695	16895	$\mathbb{F}_{2^{44}}$	384	80.90 sec	0.21 sec	118	23343	15000	$\mathbb{F}_{2^{50}}$	300	52.94 sec	0.22 sec
125	27393	17424	$\mathbb{F}_{2^{44}}$	396	80.69 sec	0.20 sec	125	30705	15488	$\mathbb{F}_{2^{44}}$	352	55.42 sec	0.16 sec

† per each ciphetext

Each element consists of 11 attributes of 40-bit entries and a leveled BGV scheme of 15 levels was used for Kim et al's protocol and 17 for ours.

λ: security parameter (bit), $\mathcal{P}$: the plaintext space of the utilized BGV scheme

m: parameter for FFT, $\phi(m)$: the dimension of a polynomial ring of the utilized BGV scheme

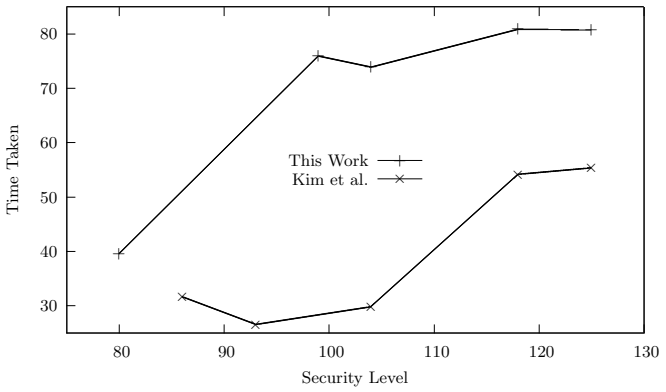

Fig. 5. Comparison of Our Protocol and Kim et al.'s [26].

Table 4 and Fig. 5 show the implementation results of running time on the server side of ours and Kim et al.'s for various security levels. Due to the difference in the number of levels required, the same m results in different security levels for ours and Kim et al.'s; this caused their scheme to be about 20-30% faster than ours. With the same security level, similar $\mathcal{P}$ and when m is

greater than 23000 for both schemes, the performance difference between the protocols is about 30-40%. Three interesting cases can be seen in the first three rows of Table 4. In the first row, with similar m, our protocol is 20% slower than Kim et al.'s protocol but due to having more plaintext slots, the amortised time of our protocol is slightly improved. In the second row, as a result of the large difference in m used in our protocol and theirs, the performance difference is almost 70%. In the third one, the larger plaintext space of $\mathbb{F}_{2^{60}}$ already makes it one of the least efficient among the experiments. Coupled with the small m of 13367 and 2 fewer levels, our performance is about 65% slower than Kim et al.'s. This highlights the difficulty in choosing m, a $\phi(m) : m$ ratio of close to 1 and having a resulting plaintext space that is suitable is rare and compromises have to be made.

The difference in performance will be slightly greater for conjunction and disjunctive queries, because the polynomial g of these two cases are much simpler than that of threshold queries. However, despite having worse performance, we remark again that the privacy of client is improved since the query type is hidden in our protocol.

6 Summary

In this work, we revisited the problem of *private database query* (PDQ), which allows a client to store its databases on a remote server in such a way that it can search over it in a private manner. We make several contributions including new security definitions and a new construction. Motivated by a subtle problem in all previous security definitions for PDQ, we pointed out that existing notions are not sufficient to cover the capabilities of attackers and propose new definitions to model better security for PDQs. Thereafter, we proposed a new PDQ protocol for conjunctive, disjunctive, and threshold conjunctive queries with equality comparison that is secure under our new definition. Finally, we provided implementation results of our PDQ protocol.

Acknowledgment

Myungsun Kim was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014-R1A1A2058377). Hyung Tae Lee, San Ling, Benjamin Hong Meng Tan, and Huaxiong Wang were supported by Research Grant TL-9014101684-01 and the Singapore Ministry of Education under Research Grant MOE2013-T2-1-041. Shu Qin Ren was supported by Research Project of Accelerated Fully Homomorphism Encryption at Data Storage Institute, A*STAR.

References

1. GMP: The GNU multiple precision arithmetic library ver. 6.1.0. Available at http://gmplib.org.
2. Amazon. Amazon relational database service. Available at https://aws.amazon.com/rds/.
3. A. Arasu and R. Kaushik. Oblivious query processing. In N. Schweikardt, V. Christophides, and V. Leroy, editors, *International Conference on Database Theory (ICDT) 2014*, pages 26–37. OpenProceedings.org, 2014.
4. F. Bao, R. H. Deng, X. Ding, and Y. Yang. Private query on encrypted data in multi-user settings. In L. Chen, Y. Mu, and W. Susilo, editors, *International Conference on Information Security Practice and Experience (ISPEC) 2008*, volume 4991 of *LNCS*, pages 71–85. Springer, 2008.
5. D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private database queries using somewhat homomorphic encryption. In M. J. J. Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *Applied Cryptography and Network Security (ACNS) 2013*, volume 7954 of *LNCS*, pages 102–118. Springer, 2013.
6. D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, 2015.

7. J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In M. Stam, editor, *IMA International Conference on Cryptography and Coding (IMACC) 2013*, volume 8308 of *LNCS*, pages 45–64. Springer, 2013.
8. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science (ITCS) 2012*, pages 309–325. ACM, 2012.
9. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *IEEE Symposium on Foundations of Computer Science (FOCS) 2011*, pages 97–106. IEEE Computer Society, 2011.
10. G. S. Çetin, Y. Doröz, B. Sunar, and E. Savas. Depth optimized efficient homomorphic sorting. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 61–80. Springer, 2015.
11. J. H. Cheon, M. Kim, and M. Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Trans. Information Forensics and Security*, 11(1):188–199, 2016.
12. CIO, Oct 7, 2014. http://www.cio.com/.
13. J. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *LNCS*, pages 487–504. Springer, 2011.
14. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security (ACM CCS) 2006*, pages 79–88. ACM, 2006.
15. L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, 2015.
16. B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *IEEE Symposium on Security and Privacy 2015*, pages 395–410. IEEE Computer Society, 2015.
17. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, 2004.
18. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *ACM Symposium on Theory of Computing (STOC) 2009*, pages 169–178. ACM, 2009.
19. O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In A. V. Aho, editor, *ACM Symposium on Theory of Computing (STOC) 1987*, pages 182–194. ACM, 1987.
20. O. Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.
21. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.
22. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, 2014.
23. S. Halevi and V. Shoup. HElib: Software library for homomorphic encryption. `http://github.com/shaih/HElib.git`, 2013.
24. B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *International Conference on Very Large Data Bases (VLDB) 2004*, pages 720–731. Morgan Kaufmann, 2004.
25. M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. In S. Jajodia and D. Wijesekera, editors, *DBSec 2005*, volume 3654 of *LNCS*, pages 325–337. Springer, 2005.
26. M. Kim, H. T. Lee, S. Ling, and H. Wang. On the efficiency of FHE-based private queries. 2016. Accepted for publication in *IEEE Trans. Dependable and Secure Computing*, Available at `http://eprint.iacr.org/2015/1176`.
27. L. Kissner and D. X. Song. Privacy-preserving set operations. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.
28. E. Kushilevitz, S. Lu, and R. Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In Y. Rabani, editor, *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 2012*, pages 143–156. SIAM, 2012.
29. Y. Lu and G. Tsudik. Privacy-preserving cloud database querying. *Journal of Internet Services and Information Security*, 1(4):5–25, 2011.
30. F. G. Olumofin and I. Goldberg. Privacy-preserving queries over relational databases. In M. J. Atallah and N. J. Hopper, editors, *Privacy Enhancing Technologies (PETS) 2010*, volume 6205 of *LNCS*, pages 75–92. Springer, 2010.

31. R. Ostrovsky. Efficient computation on oblivious RAMs. In H. Ortiz, editor, *ACM Symposium on Theory of Computing (STOC) 1990*, pages 514–523. ACM, 1990.
32. V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. Bellovin. Blind seer: A scalable private DBMS. In *IEEE Symposium on Security and Privacy 2014*, pages 359–374. IEEE Computer Society, 2014.
33. V. Pappas, M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Private search in the real world. In R. H. Zakon, J. P. McDermott, and M. E. Locasto, editors, *Annual Computer Security Applications Conference (ACSAC) 2011*, pages 83–92. ACM, 2011.
34. H.-A. Park, J. Zhan, and D. H. Lee. Privacy preserving SQL queries. In *International Conference on Information Security and Assurance (ISA) 2008*, pages 549–554. IEEE Computer Society, 2008.
35. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In T. Wobber and P. Druschel, editors, *ACM Symposium on Operating Systems Principles (SOSP) 2011*, pages 85–100. ACM, 2011.
36. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 4(11):169–180, 1978.
37. V. Shoup. NTL: A library for doing number theory version 9.7. Available at http://www.shoup.net/ntl/.
38. E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM SIGSAC Conference on Computer and Communications Security (ACM CCS) 2013*, pages 299–310. ACM, 2013.
39. S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, 2013.
40. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43. Springer, 2010.
41. Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *European Symposium on Research in Computer Security (ESORICS) 2006*, volume 4189 of *LNCS*, pages 479–495. Springer, 2006.