IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 10, 2019, accepted September 29, 2019, date of publication November 5, 2019,
date of current version November 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2951695

Search Condition-Hiding Query Evaluation on Encrypted Databases

MYUNGSUN KIM[1], HYUNG TAE LEE[2], SAN LING[3], SHU QIN REN[4], BENJAMIN HONG MENG TAN[5], AND HUAXIONG WANG[3]

[1]Department of Information Security, College of ICT Convergence, University of Suwon, Hwaseong 18323, South Korea
[2]Division of Computer Science and Engineering, College of Engineering, Jeonbuk National University, Jeonju 54896, South Korea
[3]Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371
[4]Continental Automotive Singapore Pte Ltd, Singapore 339780
[5]Institute for Infocomm Research, A*STAR, Singapore 138632

Corresponding author: Hyung Tae Lee (hyungtaelee@chonbuk.ac.kr)

The work of M. Kim was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), Ministry of Education under Grant NRF-2017R1D1A1B04035209. The work of H. T. Lee was supported by the National Research Foundation of Korea (NRF) through the Korean Government (MSIT) under Grant NRF-2018R1C1B6008476. The work of S. Ling and H. Wang was supported by the Singapore Ministry of Education under Grant MOE2016-T2-2-014(S). The work of B. H. M. Tan was supported by the Institute for Infocomm Research, A*STAR, Singapore. The work of S. Q. Ren was supported by the Research Project of Accelerated Fully Homomorphism Encryption at Data Storage Institute, A*STAR, Singapore.

ABSTRACT Private database query (PDQ) is a protocol between a client and a database server, designed for processing queries to encrypted databases. Specifically, PDQ enables a client to submit a search query and to learn a resulting set satisfying its search condition, without revealing sensitive information about a query statement. The whole query can be protected from the server, but for efficiency reasons known PDQ solutions generally consider to hide the constants only in a query statement. In this paper, we provide two fully homomorphic encryption (FHE)-based PDQ protocols that hide type of queries as well as the constants of a query statement. Particularly, our constructions focus on conjunctive, disjunctive, and threshold conjunctive queries. To this end, we first build a single compact logical expression to cover both conjunctive and disjunctive queries. On top of the logical expression, we design a PDQ protocol that enables to evaluate conjunctive and disjunctive queries without revealing any information on a given query. The second PDQ protocol comes from our observation that if a threshold conjunctive query has a particular threshold value, it results in either a conjunctive query or a disjunctive query. Because the PDQ protocol writes the three types of queries into a single polynomial expression, the resulting protocol can evaluate the three types of query statements without revealing any information on queries. To demonstrate their efficiency, we provide proof-of-concept implementation results of our proposed PDQ protocols. According to our rudimentary experiments, it takes 37.57 seconds to perform a query on 316 elements consisting of 16 attributes of 64 bits using Brakerski-Gentry-Vaikuntanathan's leveled FHE with SIMD techniques for 149-bit security, yielding an amortized rate of just 0.119 seconds per element.

INDEX TERMS Private queries, encrypted database, homomorphic encryption.

I. INTRODUCTION

Cloud computing involves highly durable storage platforms supporting a wide scope of services. One popular application is running a relational database in the cloud (e.g., AWS [1]), but it is not necessarily limited to relational databases. Many enterprises and organizations move their databases to the cloud so that they can enjoy sustainable cost advantages, robustness, and availability. However, they face the privacy problem that affects outsourcing, 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.

1) Protection of outsourced data from theft by hackers and workers on the cloud side.

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

2) Protection of submitted queries from being disclosed to the server: SQL-like query 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.

The first issue can be resolved by using encryption. In this work, we consider fully homomorphic encryption (FHE) since its capability of computing on encrypted data allows both non-aggregate (i.e., search) and aggregate query operations over encrypted databases. We can then pay more attention to answering the second privacy problem. In what follows, before formalizing the second privacy problem, we consider a specific example. Though elementary, the example illustrates the essential features and motivations of our solution(s).

A. THE PROBLEM STATEMENT

Consider the following relational algebra:

$$\pi_{\mathsf{Name}}(\sigma_{\substack{\mathsf{Major}=\text{'CS'}\wedge\mathsf{Sex}=\text{'M'}\wedge\\ \mathsf{Grade}=\text{'A'}\wedge\mathsf{Class}=\text{'DB'}}}(\mathsf{STUDENT})). \qquad (\mathcal{Q})$$

This query statement consists of a value attribute Name and four keyword attributes (e.g., Major and Grade). In addition, each keyword attribute is connected conjunctively and is checked against a specific constant value (e.g., 'A').

Given encrypted databases, the server should be able to process the query $\mathcal{Q}$ over encrypted databases. In particular, such processing is called private database query (PDQ) when it enables to produce an output without revealing any (sensitive) information of search conditions. Although using cryptographic tools such as ORAM enables to hide the whole content of query statement from servers, balancing performance against privacy must be taken into account, which is the non-trivial point in developing PDQ protocols. For this reason, existing PDQ solutions (e.g., [2], [3]) in the context of FHE mainly focus on hiding the values in a search condition. Thus such PDQ solutions consider the query $\mathcal{Q}$ in form of

$$\pi_{\mathsf{Name}}(\sigma_{\substack{\mathsf{Major}=\text{'**'}\wedge\mathsf{Sex}=\text{'**'}\wedge\\ \mathsf{Grade}=\text{'**'}\wedge\mathsf{Class}=\text{'**'}}}(\mathsf{STUDENT})).$$

However, as above, only encrypting the constants is not sufficient to address the second privacy issue. Indeed, because search conditions consist of predicates linked by the logical operators and each predicate expresses user's personal interest, it is quite desirable to *hide a way* in which a search condition uses which logical connectives. Hereafter we refer to the way as the structure of a search condition or the query structure simply. It is desirable for a PDQ protocol to handle the query $\mathcal{Q}$ in privacy-enhanced form of

$$\pi_{\mathsf{Name}}(\sigma_{\substack{\mathsf{Major}=\text{'**'}\odot\mathsf{Sex}=\text{'**'}\odot\\ \mathsf{Grade}=\text{'**'}\odot\mathsf{Class}=\text{'**'}}}(\mathsf{STUDENT})).$$

As you will see, thus the main contribution of this paper is to present an efficient algorithm to protect the structure as well as the constant values in a search condition from the server, while preserving the performance compared to existing research. We note that this work is only interested in conjunctive, disjunctive and threshold conjunctive queries with equality comparison, but our technique can be applied to PDQs with greater-than comparison (see [4]).

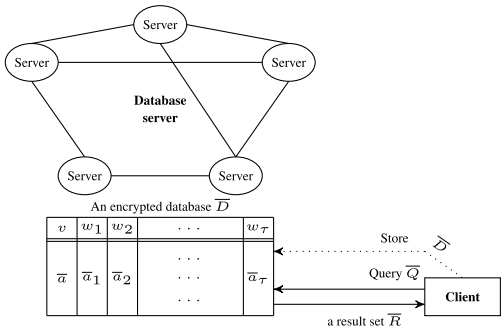


FIGURE 1. Our PDQ system model.

B. SYSTEM MODEL

The system model for PDQ we consider consists of a client and a server: The client who has his own database, stores it at the server. When a need arises, he submits a query to the server and receives a resulting set over the stored database from the server. Here, the client does not want to reveal any partial information about his queries and database to the server and/or the adversary.

To provide a PDQ solution working on the model, we consider the following scenario, shown in Fig. 1. In the initial deployment phase, the client encrypts his database as $\overline{D}$ and stores it at the cloud database server. When he sends a query request to the server, he sends a proper encryption $\overline{Q}$ of a target query statement Q. Then, the server evaluates the encrypted query with the encrypted database homomorphically and returns an encrypted result set $\overline{R}$.

Our work uses an FHE scheme to encrypt databases and query conditions. The main difference from existing works (e.g., [4], [5]) is that we try to hide not only the constant values but also the query structure.

Throughout the paper, $\overline{m}$ denotes an encryption of m for any message m. As a matter of fact, we consider only a leveled FHE scheme since it suffices for our purpose of efficiently evaluating a query for PDQ.

C. OVERVIEW OF OUR CONTRIBUTIONS

In this work we tackle the second challenge for conjunctive, disjunctive and threshold conjunctive PDQs with equality comparison. More specifically, the client would like to request the server to evaluate one of conjunctive, disjunctive and threshold conjunctive queries, but does not want to reveal any information about queries, e.g., which type of query is requested and which attributes are included in the query. To this end, we come up with a method to represent a target query in the same form, apart from query types.

TABLE 1. Results of Eq. (1) by setting constants.

Query Type	b	c	d	Result of Eq. (1)
Conjunction	1	0	0	$0/1$
Disjunction	1	1	1	$0/1$

1) OUR KEY IDEAS

We present two PDQ protocols that hide query structures as well as constants in query statements. One is a protocol that evaluates conjunctive and disjunctive queries with equality comparison and the other extends the type of evaluable queries to threshold conjunctive queries. Now, we first look into our key observations, which are used for designing our PDQ protocols.

1) We first observe that conjunction and disjunction with equality comparison can be represented with the same circuit structure by using an equality test circuit. Let EQ 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 consider a circuit

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

to represent conjunctive and disjunctive queries with equality comparison between A_i and a_i for each $1 \le i \le \tau$. Then, the results of the circuit in Eq. (1) are determined as in Table 1 with respect to queries and the choice of constants b, c, and d. We provide a new design of a PDQ protocol that evaluates conjunctive and disjunctive queries with equality comparison using Eq. (1).

2) Our next observation is that conjunctive and disjunctive queries are special forms of threshold conjunctive queries. More specifically, a conjunctive query is a threshold conjunctive query with threshold $T = \tau$ and a disjunctive query is a threshold conjunctive query with threshold $T = 1$. Thus, once if a threshold value is hidden to the server, we can hide where a given query lies among conjunctive, disjunctive and threshold conjunctive queries.
To achieve this, the client sets the constants in Eq. (1) as follows:

$$b = 1 + t, \quad c = 0 \text{ and } d = 1$$

where t is a proper element in a multiplicative group $\mathbb{F}^*_{2^\ell}$. [1] Then, the evaluation result of Eq. (1) amounts to t^κ where κ is the number of elements such that $A_i = a_i$ for $1 \le i \le \tau$. (Note that $1 + (1 + t) = t$ since the characteristic of $\mathbb{F}_{2^\ell}$ is 2.) Then the client only has to let the server evaluate an encrypted polynomial $\overline{g}$ such that $g(t^\kappa) = 1$ if $T \le \kappa \le \tau$ and 0 if $0 \le \kappa < T$ for a threshold T.

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

At the end of the steps, the server has the intermediate results of the query, i.e., $\overline{1}$ for true and $\overline{0}$ for false. Therefore, he can obtain the final result by multiplying the encrypted data (the values in the v column in Fig. 1) and the intermediate results.
Based on the observation, we provide a new PDQ protocol that evaluates conjunctive, disjunctive and threshold conjunctive queries with equality comparison, without revealing the query structure as well as the constants in a query statement.

2) OUR RESULTS

The contributions of this work are two-fold:

- *New privacy-enhanced PDQ protocols.* We devise two PDQ protocols over encrypted databases with the following properties. The first protocol evaluates conjunctive and disjunctive protocols and requires a total multiplicative depth of $\lceil \log \ell \rceil + \lceil \log \tau \rceil + 2$ 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 costs $(2\tau + 1)$ FHE ciphertexts from the client and n FHE ciphertexts from the server where n is the number of tuples in the database. The second protocol can additionally evaluate threshold conjunctive queries at the costs of $\lceil \log(1 + \tau) \rceil$ multiplicative depth for computation and τ FHE ciphertexts from the client for communication.
- *Experimental study.* We present implementation results of our designs of PDQ protocols. Because our protocols are computationally intensive, we apply some additional techniques to improve the performance of the protocols. 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. Further, we give a comparison of ours with the previous standard query over encrypted data using FHE in [5], that achieves weaker privacy than ours. From our experiments, it requires 37.57 seconds to perform a query on 316 elements consisting of 16 attributes of 64 bits using Brakerski-Gentry-Vaikuntanathan (BGV) leveled FHE with SIMD techniques for 149-bit security, yielding an amortized rate of just 0.119 seconds per element.

D. RELATED WORK

A popular PDQ system CryptDB [6] and its offshoot Monomi [7] 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 [4], [5], [8]–[10] based on SHE and FHE. Boneh et al. [8] 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. [4] 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 $\{0, 1\}$. Subsequently, Saha and Koshiba [9] presented an improved PDQ protocol for conjunctive queries by proposing a new packing method. In [10], Saha et al. also provided PDQ protocols for conjunctive and disjunctive queries and they considered to hide attribute information as well as constants in query statements.

Very recently, Kim et al. [5] 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 [8], [11], [12]. However, the works mentioned above are not also concerned with hiding the type of query that is evaluated.

For hiding the type of query, Goldwasser et al. [13] 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. [14] constructed a secret-key MI-FE that is more efficient but they are not practical yet.

A more practical approach to hiding queries is [15] 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 server 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], [17], 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.

E. THE OUTLINE OF THE PAPER

Our paper is structured in the following way. In Section II, we introduce a structure of database and types of queries that we handle in this paper, and recall the concept of FHE and recent results on the required depth for an equality test algorithm using FHE. Our two PDQ protocols are presented in Sections III and IV, respectively, followed by subsections that show an in-depth performance evaluation and analyze the security each. In Section V, we provide our proof-of-concept implementation and extensive experimental results with various parameters.

II. PRELIMINARIES

In this section, we first introduce a database structure and types of queries that we consider throughout this paper. Then, we briefly review the concept of FHE and look over the recent results on the multiplicative depth to perform an equality test algorithm between FHE-encrypted data.

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 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 $a \in \mathbb{R}$.

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 $\mathsf{poly}(\lambda)$.

A. DATABASE MODEL

For better understanding, we need to introduce some notation for databases and queries along with their types and structures.

1) DATABASE STRUCTURE

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, the element $\alpha_i.v$ represents a value attribute (e.g., student ID) which is retrieved if the tuple α_i satisfies the search condition. In contrast, each attribute $\alpha_i.w_j$ represents a keyword attribute (e.g., his grade) which is checked against the conditions in a search (e.g., a `where`-clause in SQL). Lastly, we denote by $\boldsymbol{R}_D(Q)$ a result set from executing a query Q over a database D.

For simplicity, we assume that the schema has one value attribute (and note that it is straightforward to expand it to multiple value attributes).

2) QUERY, ITS TYPE AND STRUCTURE

Throughout the paper, we consider the following three types of queries.

- Conjunctive query (CQ): A query to return v_i's such that $\bigwedge_{j\in J}(\alpha_i.w_j = a_j)$.
- Disjunctive query (DQ): A query to return v_i's such that $\bigvee_{j\in J}(\alpha_i.w_j = a_j)$.
- Threshold conjunctive query (TCQ): A query to return v_i's such that $\kappa \geq T$ where T is a positive integer and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$.

For notational simplicity, any query in the above three types can be written as the following pair of tuples,

$$\left((\odot, J, T), \{a_j\}_{j\in J}\right) \qquad (\dagger)$$

for $\odot \in \{\wedge, \vee, \wedge^T\}$, an index set $J \subseteq [\tau]$ and a threshold T where $\wedge$, $\vee$ and $\wedge^T$ indicate CQ, DQ and TCQ, respectively. We note that a CQ is parameterized with $T = |J|$ but a DQ with $T = 1$.

We call the first component $(\odot, J, T)$ in (†) a *query structure* and the second component $\{a_j\}_{j\in J}$ a *query constant*, sometimes constant simply.

B. FULLY HOMOMORPHIC ENCRYPTION

A fully homomorphic encryption (FHE) scheme consists of the following four PPT algorithms, KG, Enc, Dec, and Eval:

- $\mathsf{KG}(\lambda) \to (pk, ek, sk)$: 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 $\mathcal{C}$.
- $\mathsf{Enc}(pk, m) \to \overline{m}$: Given the public key pk and a plaintext message $m \in \mathcal{P}$, it outputs a ciphertext $\overline{m}$.
- $\mathsf{Dec}(sk, \overline{m}) \to m^*/\ \perp$: Given the secret key sk and a ciphertext $\overline{m}$, it outputs a message $m^* \in \mathcal{P}$ or $\perp$.
- $\mathsf{Eval}(ek, \varphi, \overline{m}_1, \ldots, \overline{m}_n) \to \overline{m}_\varphi$: It takes the evaluation key ek, a function $\varphi : \mathcal{P}^n \to \mathcal{P}$, and a set of n ciphertexts $\overline{m}_1, \ldots, \overline{m}_n$ 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 attacks. 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 PPT 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 $\mathsf{KG}(1^\lambda) \to (pk, ek, sk)$ and $m_0, m_1 \in \mathcal{P}$ are chosen by the adversary $\mathcal{A}$.

Since Rivest et al. [18] 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 [19]. Following his design philosophy, various studies [20]–[22] have been presented on constructing efficient FHE schemes, however they have fairly poor performance.

To resolve the efficiency problem, Brakerski and Vaikuntanathan [23] introduced the concept of leveled FHE schemes, which allows the evaluation of functions of at most a predetermined multiplicative depth, instead of arbitrary functions. Shortly after, Brakerski, Gentry, and Vaikuntanathan (BGV) [24] 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 [24], [25] in our proposed protocol.

C. EQUALITY TEST ON ENCRYPTIONS

Our protocols 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 $\mathsf{EQ}(\cdot, \cdot)$, is defined as follows: For two messages $m_1, m_2 \in \mathcal{P}$,

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

As mentioned above, its computational efficiency is also determined by the required multiplicative depth when evaluated using that FHE scheme. Some works in the literature [4], [5], [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. [5], 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 proposed protocols will utilize FHE schemes, such as [24], [25], whose plaintext space is $\mathbb{F}_{2^\ell}$ and which consumes no multiplicative depth to evaluate the Frobenius map.

III. OUR SEARCH CONDITION-HIDING PDQ PROTOCOL

In this section, we describe our search condition-hiding private database query (SCH-PDQ) protocol that hides whether a query is either a CQ or a DQ. Subsequently, we look into the correctness, the efficiency, and the security of the proposed SCH-PDQ protocol.

A. WARM-UP

Before describing our SCH-PDQ protocol, we first give some notations and assumptions used for our protocols in this and the next sections.

We assume an encrypted database $\overline{D} = \overline{\alpha}_1 \| \overline{\alpha}_2 \| \cdots \| \overline{\alpha}_n$ using an FHE scheme is given to the server, where $\alpha_i = (\alpha_i.v, \alpha_i.w_1, \ldots, \alpha_i.w_\tau)$ 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., $\overline{\alpha}_i = (\alpha_i.\overline{v}, \alpha_i.\overline{w}_1, \ldots, \alpha_i.\overline{w}_\tau)$. For convenience, we frequently use v_i and w_{ij} in place of $\alpha_i.v$ and $\alpha_i.w_j$, respectively. [2]

For the correctness of our protocols, 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 protocols.

For the purpose of 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 [24], [25]. In this case, the EQ algorithm that is a main component of our protocols, consumes $\lceil \log \ell \rceil$ multiplicative depth from the analysis in [5]. Throughout the paper, $p(x)$ denotes an irreducible polynomial of degree ℓ where $\mathbb{F}_{2^\ell}$ is isomorphic to $\mathbb{F}_2[x]/\langle p(x) \rangle$. We denote by $t \in \mathbb{F}_{2^\ell}$ a root of $p(x)$ and it is predetermined before beginning the protocols.

To encrypt an ℓ-bit message $a = (a_\ell, \ldots, a_1) \in \{0,1\}^\ell$ 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 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 notice that $\overline{1} + \overline{1} = \overline{0}$ since it is assumed that the plaintext space of the exploited FHE scheme in our protocols is $\mathbb{F}_{2^\ell}$.

B. OUR SCH-PDQ PROTOCOL DESCRIPTION

We now give the detailed description of our SCH-PDQ protocol. Recall the two types of queries–CQ and DQ.

- A conjunctive query (CQ) means a query to return v_i's such that $\bigwedge_{j \in J} (\alpha_i.w_j = a_j)$.
- A disjunctive query (DQ) means a query to return v_i's such that $\bigvee_{j \in J} (\alpha_i.w_j = a_j)$.

Throughout the paper, we refer to J as an index set and $J \subseteq [\tau]$.

The Description: The following is the description of our SCH-PDQ protocol between the client and the server.

[2] It is assumed that the client and the server share the schema description of D as well.

1) The client performs as follows: On input query Q with the index set J and constants a_j for $j \in J$,
   a) For all $j \in [\tau]$, set
      - $b_j = \begin{cases} 1 & j \in J \\ 0 & \text{otherwise,} \end{cases}$ and
      - $a_j \xleftarrow{\$} \{0,1\}^\ell$ if $j \notin J$.
   b) Set $c = \begin{cases} 1 & \text{if } Q \text{ is conjunction} \\ 0 & \text{if } Q \text{ is disjunction} \end{cases}$
   c) Encrypt a_j, b_j for all $j \in [\tau]$ and c, and send them to the server.
2) The server performs as follows:
   a) Compute $\overline{d}_j = \overline{1} + \overline{c} \cdot \overline{b}_j$ for all $j \in [\tau]$.
   b) Compute
   $$\overline{\beta}_{ij} = \overline{d}_j + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{b}_j \tag{2}$$
   for all $i \in [n]$ and $j \in [\tau]$.
   c) Compute $\overline{\zeta}_i = (\overline{1} + \overline{c}) + \prod_{j \in [\tau]} \overline{\beta}_{ij}$ for all $i \in [n]$.
   d) Compute $\overline{\gamma}_i = \overline{\zeta}_i \cdot \overline{v}_i$ for all $i \in [n]$.
   e) Send $\{\overline{\gamma}_1, \cdots, \overline{\gamma}_n\}$ to the client.
3) The client obtains $\gamma_1, \cdots, \gamma_n$ by decrypting $\overline{\gamma}_i$'s using the secret key of the exploited FHE scheme.

C. ANALYSIS OF OUR SCH-PDQ PROTOCOL

We begin with discussing why the SCH-PDQ protocol works correctly and then analyze its costs in terms of computation and communication. The security property of our construction will be studied in the next subsection.

1) CORRECTNESS

The following theorem argues the correctness of our SCH-PDQ protocol.

Theorem 1: Let Q be a query statement over a database D such that it is either a CQ or a DQ. Then, the client obtains the result set $\boldsymbol{R}_D(Q)$ correctly after the execution of our SCH-PDQ protocol.

Proof: Our proof proceeds in a case-by-case manner.

1) Conjunction ($\overline{c} = \overline{1}$): Let Q be a CQ. Then,
   - If $j \in J$, $\overline{b}_j = \overline{1}$ meaning $\overline{d}_j = \overline{1} + \overline{c} \cdot \overline{b}_j = \overline{0}$ and
   $$\begin{aligned} \overline{\beta}_{ij} &= \overline{d}_j + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{b}_j \\ &= \overline{0} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{1} \\ &= \begin{cases} \overline{1} & \text{if } \alpha_i.w_j = a_j \\ \overline{0} & \text{if } \alpha_i.w_j \neq a_j. \end{cases} \end{aligned}$$
   - If $j \neq J$, $\overline{b}_j = \overline{0}$ meaning $\overline{d}_j = \overline{1} + \overline{c} \cdot \overline{b}_j = \overline{1}$ and
   $$\begin{aligned} \overline{\beta}_{ij} &= \overline{d}_j + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{b}_j \\ &= \overline{1} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{0} = \overline{1}. \end{aligned}$$

Let $\overline{c}' = \overline{1} + \overline{c}$. Then, for each $i \in [n]$,

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

and

$$\overline{\gamma}_i = \overline{\zeta}_i \cdot \overline{v}_i = \begin{cases} \overline{v}_i & \text{if } \alpha_i.w_j = a_j \text{ for all } j \in J \\ \overline{0} & \text{otherwise.} \end{cases}$$

2) Disjunction ($\overline{c} = \overline{0}$): In turn, let Q be a DQ. Then,

- If $j \in J$, $\overline{b}_j = \overline{1}$ meaning $\overline{d}_j = \overline{1} + \overline{c} \cdot \overline{b}_j = \overline{1}$ and

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

- If $j \neq J$, $\overline{b}_j = 0$ meaning $\overline{d}_j = \overline{1} + \overline{c} \cdot \overline{b}_j = \overline{1}$ and

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

Similarly, let $\overline{c}' = \overline{1} + \overline{c}$. Then, for each $i \in [n]$,

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

and

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

From the above, we see 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. □

In what follows, we analyze the efficiency of our SCH-PDQ protocol.

2) COMPUTATIONAL COSTS

We evaluate the computational cost of our SCH-PDQ protocol in terms of the required multiplicative depth. At Step 2) b), it requires $\lceil \log \ell \rceil + 1$ multiplicative depth to compute $\overline{\beta}_{ij}$ because the EQ 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 [5]. (Note that 1 multiplicative depth for Step 2) a) is not counted since it is dominated by the second term of the right side in Eq. (2).) At Step 2) c), it takes $\lceil \log \tau \rceil$ multiplicative depth to compute $\prod_{j\in[\tau]} \overline{\beta}_{ij}$. Computing $\overline{\gamma}_i$ at Step 2) d) consumes 1 multiplicative depth. Thus, the total required multiplicative depth is $\lceil \log \ell \rceil + \lceil \log \tau \rceil + 2$.

3) COMMUNICATION COSTS

At Step 1) c), the client sends encryptions of a_j, b_j and c for all $j \in [\tau]$. Hence, the communication costs of the client come to $(2\tau + 1)$ FHE-ciphertexts. On the other hand, the server sends $\overline{\gamma}_i$'s for all $i \in [n]$ to the client at Step 2) e). Hence, the communication costs of the server come to n FHE-ciphertexts.

D. SECURITY OF OUR SCH-PDQ PROTOCOL

In this section, we examine the security property of our SCH-PDQ construction. We will argue that the construction of our SCH-PDQ protocol in Section III-B hides the query structure as well as the query constant in a query statement Q, which implies that we can enhance the privacy of PDQ protocols.

Theorem 2: Assume that the exploited FHE scheme is IND-CPA secure, then in the semi-honest adversary model, no PPT algorithms can learn any information about the query structure and the constants of a query statement which is either a CQ or a DQ.

Proof: It suffices to build a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, given a set H of polynomially many encrypted queries over an encrypted database $\overline{D}$ by the exploited FHE scheme, $\mathcal{S}$ can simulate the adversary's view $\mathcal{V}$. 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 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

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

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

- Generating $\overline{D}^*$. $\mathcal{S}$ first initializes $\overline{D}^* \leftarrow \varnothing$. For each $\alpha_{i\in[n]}^* \xleftarrow{\$} \mathcal{P}^{(\tau+1)}$ where $\mathcal{P}$ is the plaintext space of the exploited FHE scheme, 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 $\overline{Q}^*$. From the list H 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 $\overline{a}_j^*$ and $\overline{b}_j^*$ with $a_j^* \xleftarrow{\$} \mathcal{P}$ and $b_j^* \xleftarrow{\$} \mathcal{P}$ using the FHE encryption algorithm under the public key pk in the same way as above. Similarly, it computes $\overline{c}^*$ under the public key pk for $c^* \xleftarrow{\$} \mathcal{P}$.
- Generating $\boldsymbol{R}_{\overline{D}^*}(\overline{Q}^*)$. 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 $\boldsymbol{R}_{\overline{D}^*}(\overline{Q}^*)$.

We show that $\mathcal{V}^*$ is computationally indistinguishable from $\mathcal{V}$ by comparing them component-wise. 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 $\overline{Q}$ and a simulated query $\overline{Q}^*$. Then we see that

$$\{\overline{a}_j, \overline{b}_j\}_{j\in[\tau]} \overset{c}{\approx} \{\overline{a}_j^*, \overline{b}_j^*\}_{j\in[\tau]} \text{ and } \overline{c} \overset{c}{\approx} \overline{c}^*$$

if the exploited FHE scheme is IND-CPA secure, where $\overset{c}{\approx}$ means computational indistinguishability.

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. □

IV. OUR THRESHOLD SCH-PDQ PROTOCOL

In this section, we provide our threshold search condition-hiding private database query (TSCH-PDQ) protocol that hides which of the three query types–CQ, DQ, and TCQ a query lies in.

Recall that a TCQ means a query to return v_i's such that $\kappa \geq T$ where T is a pre-fixed positive integer and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$. We observe that CQs and DQs are special forms of TCQs. More concretely,

- A CQ, which returns v_i's such that $\bigwedge_{j\in J}(\alpha_i.w_j = a_j)$, can be interpreted as a TCQ with threshold $T = |J|$.
- A DQ, which returns v_i's such that $\bigvee_{j\in J}(\alpha_i.w_j = a_j)$, can be also interpreted as a TCQ with threshold $T = 1$.

This is the reason why our TSCH-PDQ protocol attempts to evaluate a function with concealing a threshold value T from the server.

A. OUR TSCH-PDQ PROTOCOL DESCRIPTION

The Description: We now give the description of our TSCH-PDQ protocol below.

1) The client performs as follows: On input a TCQ Q which requests to return v_i's such that $\kappa \geq T$ for a pre-determined positive integer T and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$,
   a) For all $j \in [\tau]$, set
      - $b_j = \begin{cases} 1+t & j \in J \\ 0 & \text{otherwise} \end{cases}$ and
      - $a_j \xleftarrow{\$} \{0,1\}^\ell$ if $j \notin J$.
   b) Compute a polynomial $g = \sum_{k=0}^{\tau} g_k x^k \in \mathbb{F}_{2^\ell}[x]$ such that $g(t^\kappa) = \begin{cases} 1 & \text{if } T \leq \kappa \leq \tau \\ 0 & \text{if } 0 \leq \kappa < T. \end{cases}$
   c) Encrypt a_j, b_j for all $j \in [\tau]$ and g_k for all $0 \leq k \leq \tau$, and send them to the server.
2) The server performs as follows:
   a) Compute
   $$\overline{\beta}_{ij} = \overline{1} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{b}_j \tag{3}$$
   for all $i \in [n]$ and $j \in [\tau]$.
   b) Compute $\overline{\zeta}_i = \prod_{j\in[\tau]} \overline{\beta}_{ij}$ for all $i \in [n]$.
   c) Compute $\overline{\gamma}_i = g(\overline{\zeta}_i) \cdot \overline{v}_i$ for all $i \in [n]$.
   d) Send $\{\overline{\gamma}_1, \ldots, \overline{\gamma}_n\}$ to the client.
3) The client obtains $\gamma_1, \ldots, \gamma_n$ by decrypting $\overline{\gamma}_i$'s using the secret key of the exploited FHE scheme.

B. ANALYSIS OF OUR TSCH-PDQ PROTOCOL

1) CORRECTNESS

The following theorem shows the correctness of our TSCH-PDQ protocol.

Theorem 3: For a pre-fixed positive integer T and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$, let Q be a TCQ over a database D that returns v_i's such that $\kappa \geq T$. Then, the client obtains the result set $\boldsymbol{R}_D(Q)$ correctly after the execution of our TSCH-PDQ protocol.

Proof: Let Q be a TCQ that returns v_i's such that $\kappa \geq T$ where T is a pre-determined positive integer and $\kappa = |\{j \in J | \alpha_i.w_j = a_j\}|$. Then,

- If $j \in J$,
$$\begin{aligned} \overline{\beta}_{ij} &= \overline{1} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{b}_j \\ &= \overline{1} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{1+t} \\ &= \begin{cases} \overline{t} & \text{if } \alpha_i.w_j = a_j \\ \overline{1} & \text{if } \alpha_i.w_j \neq a_j. \end{cases} \end{aligned}$$
- If $j \neq J$,
$$\begin{aligned} \overline{\beta}_{ij} &= \overline{1} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{b}_j \\ &= \overline{1} + \mathsf{EQ}(\alpha_i.\overline{w}_j, \overline{a}_j) \cdot \overline{0} = \overline{1}. \end{aligned}$$

Hence, for each $i \in [n]$,

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

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

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

by the definition of g. From the above, we confirm 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. □

2) GENERALIZING TSCH-PDQ

With our TSCH-PDQ protocol, we achieve more than hiding the threshold value by evaluating an encryption of the polynomial g. To compute threshold conditions, Kim et al. [5] used a polynomial g such that if the exponent of t in $\zeta_i = t^\kappa$ where $\kappa \geq T$ for some threshold T, then $g(\zeta_i) = 1$ and $g(\zeta_i) = 0$ otherwise. But we note that their idea is actually much more powerful.

This technique can be used to partition the set of possible outputs, $\{1, t, \ldots, t^\tau\}$. In the case for these queries, we are interested in partitioning it into two disjoint sets, T_0 and T_1 such that if $\kappa \in T_i \subseteq [\tau]$, then $g(t^\kappa) = i$ for $i \in \{0, 1\}$.

Encompassed under this framework are queries that are the inverse of threshold conditions, $g(t^\kappa) = 1$ if $\kappa \leq T$, "range" conditions, where $g(t^\kappa) = 1$ if $L \leq \kappa \leq U$ and multiple disjoint range conditions between 0 and τ.

Next, we analyze the computational and communicational costs of our TSCH-PDQ protocol.

3) COMPUTATIONAL COSTS

We first evaluate the computational costs of our TSCH-PDQ protocol in terms of the required multiplicative depth. We note at the server-side that Steps 2) a), b) and c) were slightly modified from our SCH-PDQ protocol, but Steps 2) a) and b) still consume the same amount of multiplicative depth as Steps 2) b) and c) of our SCH-PDQ scheme, respectively. On the other hand, at Step 2) c), it takes $\lceil \log(1+\tau) \rceil$ multiplicative depth additionally for evaluating a polynomial g of degree τ. As a result, the total multiplicative depth is approximately $\lceil \log \ell \rceil + 2\lceil \log(1+\tau) \rceil + 2$.

4) COMMUNICATION COSTS

Compared to our SCH-PDQ protocol, while the client does not send encryption of c, he sends encryption of all g_k's additionally, and the communication costs from the client is τ more than our SCH-PDQ protocol, i.e., $(3\tau + 1)$ FHE-ciphertexts. The communication costs from the server are exactly the same as those of our SCH-PDQ protocol, i.e., n FHE-ciphertexts.

C. SECURITY OF OUR TSCH-PDQ PROTOCOL

As before, we explore the security of our TSCH-PDQ protocol. The following theorem argues that our TSCH-PDQ protocol hides the query structure as well as the constants in a query statement Q which is one of CQ, DQ and TCQ.

Theorem 4: Assuming that the utilized FHE scheme is IND-CPA secure, no PPT algorithms can reveal any information about the query structure and the constants in a query statement which is one of CQ, DQ and TCQ, in the semi-honest adversary model.

Proof: The proof of this theorem is almost the same as that of Theorem 2, except for the way to generate $\overline{Q}^*$ since our TSCH-PDQ protocol generates a polynomial g, instead of constant c. As in the proof of Theorem 2, we build a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, given a set H of polynomially many encrypted queries over an encrypted database $\overline{D}$ by the utilized FHE scheme, $\mathcal{S}$ can simulate the adversary's view $\mathcal{V}$. It is assumed that $\overline{D}$ is already in place and the adversary can see the generation of $\overline{D}$, but it cannot reveal any extra information except for the size of $\overline{D}$ (i.e., $n = |\overline{D}|$) since the utilized FHE scheme is IND-CPA secure.

Now, we show that $\mathcal{S}$ taking as input H can generate a view $\mathcal{V}^*$ which is computationally indistinguishable from $\mathcal{V}$ where

$$\mathcal{V}^* := \left(\overline{D}^* = \{\overline{\alpha}_i^*\}_{i\in[n]}, \overline{Q}^*, \boldsymbol{R}_{\overline{D}^*}(\overline{Q}^*)\right)$$

and $\mathcal{V}$ is the real view of the adversary $\mathcal{A}$.

Then, $\mathcal{S}$ constructs $\mathcal{V}^*$ as follows. We note that the ways to generate $\overline{D}^*$ and $\boldsymbol{R}_{\overline{D}^*}(\overline{Q}^*)$ are exactly the same as those of $\mathcal{S}$ in the proof of Theorem 2.

- Generating $\overline{D}^*$. $\mathcal{S}$ first initializes $\overline{D}^* \leftarrow \varnothing$. For each $\alpha_{i\in[n]}^* \xleftarrow{\$} \mathcal{P}^{(\tau+1)}$ where $\mathcal{P}$ is the plaintext space of the utilized FHE scheme, it computes $\overline{\alpha}_i^*$ with the public key pk and a randomness r_i^*, and set $\overline{D}^* \leftarrow \overline{D}^* \cup \{\overline{\alpha}_i^*\}$.
- Generating $\overline{Q}^*$. It generates $\overline{a}_j^*$ and $\overline{b}_j^*$ for all $j \in [\tau]$. Then, 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 $\boldsymbol{R}_{\overline{D}^*}(\overline{Q}^*)$. 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 $\boldsymbol{R}_{\overline{D}^*}(\overline{Q}^*)$.

We can easily confirm that

$$\{\overline{a}_j, \overline{b}_j\}_{j\in[\tau]} \overset{c}{\approx} \{\overline{a}_j^*, \overline{b}_j^*\}_{j\in[\tau]}$$

and

$$\{\overline{g}_k\}_{k\in[\tau]} \overset{c}{\approx} \{\overline{g}_k^*\}_{k\in[\tau]}$$

if the utilized FHE scheme is IND-CPA secure, where $\overset{c}{\approx}$ means computational indistinguishability. Furthermore, 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. Therefore, we can conclude the proof of Theorem 4.

□

V. IMPLEMENTATION

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

A. EXPERIMENT SETTING

1) TEST ENVIRONMENT

The testing platform was a server with Intel® Xeon™ Platinum 8170 with a peak frequency of 3.7 GHz and 192 GB RAM. The implementation of our protocol was written in C++ using the following libraries: HElib (commit d0a00be) [27], NTL 11.3.2 [28] (AVX-accelerated FFT), and GMP Library 6.1.2 [29]. Experiments were conducted using programs running on a single thread, with the thread boosting functionalities of NTL and HElib turned off.

2) DEPTH AND LEVELS VERSUS CAPACITY

With HElib 1.0.0, the concept of ciphertext capacity is introduced to replace the notions of depth and level. Previously, ciphertexts had moduli $q_c = \prod_{i=0}^{c} p_i$, where each p_i was a 22-25 bit long prime number. After each ciphertext multiplication, the last prime p_c would be dropped and this gave an almost one-to-one correspondence between circuit depth and level; although SIMD operations such as shifts, rotations and Frobenius maps evaluations affected the actual noise of ciphertexts as well.

TABLE 2. Performance result of the condition-hiding PDQ protocols.

			Time									Communication	
			C	S								C→S	S→C
Ptcl Type	HElib Capacity	k, τ, # Packed	Step 1	16 EQ	Compute					Total	Amort.	Query	Result (Amort.)
					$\overline{\beta}_{ij}$	$\overline{d_i}$	$\overline{\zeta}_i$	$g(\overline{\zeta}_i)$	$\times v_i$				
SCH	250	48, 16, 360	5.71	24.09	1.69	2.28	1.59	—	0.064	29.71	0.083	57.0 MB	288 KB (0.8 KB)
TSCH	300		8.92	30.39	2.04	—	2.00	0.085	2.11	36.62	0.102	98.8 MB	
SCH	250	64, 16, 316	62.39	24.46	1.67	2.39	1.62	—	0.062	30.20	0.096	98.2 MB	334 KB (1.06 KB)
TSCH	310		94.48	30.80	2.00	—	2.10	2.60	0.075	37.57	0.119	114.5 MB	

One ciphertext worth of records were generated for this experiment. Timings are averages of 100 runs and given in seconds.

C: Client, S: Server

Noise growth is actually not the same for every multiplication and ciphertext operation, and depends on other factors such as plaintext modulus and current ciphertext noise levels too. Ciphertext capacity gives more fine-grained control over noise growth by allowing us to reduce the modulus by arbitrary amounts tailored to the situation. In Table 2, we mention the capacity parameters used to instantiate the BGV scheme in HElib; the actual capacity obtained will be slightly larger.

3) PARAMETER SELECTION FOR BGV SCHEME

In our experiments, we chose a plaintext space for the BGV scheme that is slightly larger than needed. This is because the parameters of the scheme are determined by the desired security level, required depth and minimum plaintext size, which greatly limit the set of suitable parameters.

Particularly, for the experiments in this work, we used a variety of parameters, with fixed plaintext space $p = 2$, key-switching digit parameter $c = 3$ and

- cyclotomic polynomial $\Phi_{20857}(X)$, yielding a plaintext space of $\mathbb{Z}_2[X]/\Phi_{20857}(X) \cong \prod_{i=1}^{316} \mathbb{F}_{2^{66}}$, allowing us to fit 64-bit entries and $\lambda \geq 149$ with the largest BGV instance used;
- cyclotomic polynomial $\Phi_{19811}(X)$, yielding a plaintext space of $\mathbb{Z}_2[X]/\Phi_{19811}(X) \cong \prod_{i=1}^{360} \mathbb{F}_{2^{50}}$, allowing us to fit 48-bit entries and $\lambda \geq 127$ with the largest BGV instance used.

4) DATABASE CHARACTERISTICS

We generated random databases, each with one value and $\tau = 16$ keyword attributes, with entries that have maximum sizes of $k = 48$ or 64 bits. Each database was tailored to fit in a single ciphertext with SIMD packing techniques to determine how the protocols would scale with such optimizations.

B. OPTIMIZATIONS

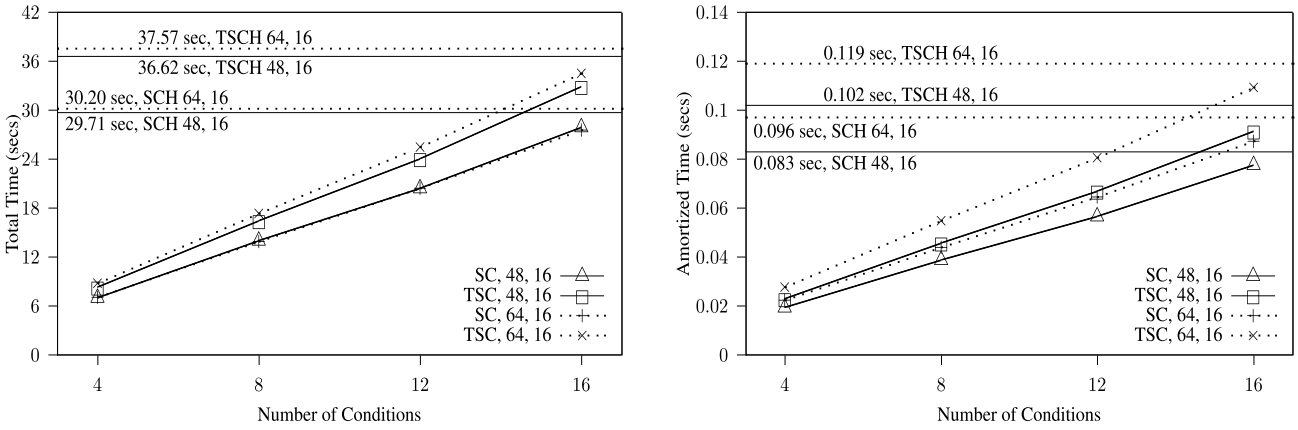
In our implementation, we employed the BGV encryption scheme [24] as the underlying FHE scheme and exploited two additional techniques to improve the performance of the protocol.

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 $x^j = \prod_{i=1}^{\lceil \log j \rceil} x^{b_i \cdot 2^i}$ 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 $0 \leq i \leq \lceil \log j \rceil$. 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 depth-reduced polynomial evaluation requires. We first evaluate all monomials whose exponent is a power of two, i.e., $x^2, x^4, \cdots$, with depth-free Frobenius automorphisms. Then, we compute in ascending order, starting with x^3, all other monomials x^i by expressing $i = e_1 + 2^j e_2$ where e_1, e_2 have similar Hamming weights and multiplying x^{e_1} and $x^{2^j e_2}$. Finally, polynomial evaluation is performed by multiplying each monomial with its corresponding coefficient and summing them up.

C. EXPERIMENTAL RESULTS

To evaluate the performance of our condition-hiding PDQ protocols, we ran them with two different databases, where entries have 48/64-bit length and present the results in Table 2 in several scenarios.

First, we determined and tested them with the best HElib parameters that achieve minimum security levels, λ, of greater than 120 bits. From the results, it is clear that the bulk of the protocol's computation time, at least 80%, is on the equality conditions. In these cases, for both entry sizes, there is an overhead of around 24% for the TSCH-PDQ protocol compared to simply hiding conjunction/disjunction. This is mainly because a larger capacity is necessary to evaluate threshold conjunction queries compared to conjunction/disjunction queries, which stems from its more complex



(a) Total Time (b) Amortized Time

FIGURE 2. Performance of various PDQ protocols.

algorithm. For the chosen entry sizes of 48 and 64, the depth required to compute the protocols does not differ theoretically which is reflected in the similar capacities used. Although we gave HElib capacities of 300 and 310 for the 48 and 64-bit TSCH-PDQ protocols, the actual capacity obtained was 313 and 320 which only has a 7-bit difference. The size of the resulting ciphertext encrypting γ_i's were quite small, less than 350 KB overall which means that the actual space overhead is about $\frac{800}{6} \approx \frac{1060}{8} \approx 133\times$ in both instances.

While the total running times of SCH-PDQ protocol were nearly indistinguishable, their amortized performance showed a gap of 0.013 seconds which is a 15% increase. This is mostly due to the greater number, around 13% more, of entries that we could pack in a single ciphertext. With greater database sizes, the amortized time of the SCH-PDQ protocols will improve slightly as the $\overline{d}_i$'s are only computed once, requiring around 2.4 seconds, and will have less impact on the overall running time.

1) IMPROVING QUERY GENERATION WITH SIMD

From Table 2, it is clear that the time and space complexity of query generation can be improved. For 64-bit entries, it took more than 1 minute to prepare the necessary data which amounted to more than 100 MB. This was because encryption required almost 2 seconds per ciphertext, and we needed to encrypt 33 and 49 plaintexts for the SCH and TSCH protocols respectively. The situation is better for 48-bit entries due to much faster encryption times but the size of the data to be sent to the server was still over 50 MB.

We can exploit SIMD techniques to improve the situation. BGV instances can actually encrypt more than 300 plaintexts into a single ciphertext. Plaintexts needed by the server for multiple queries can be encrypted into a single ciphertext, reducing communication costs to around 1 MB of data. However, the server has to process this packed ciphertext to extract the plaintexts for each query. It would incur an one-time processing overhead and increase the depth of the computation slightly.

More concretely, below we introduce an optimized query processing technique for packing the information required to execute a single query, focusing on the TSCH protocol. The client packs the query information as follows: First, let $\mathbf{v}$ be the vector of the a_j's, b_j's and coefficients of g, $(a_1, \ldots, a_\tau, b_1, \ldots, b_\tau, g_0, \ldots, g_\tau, 0, \ldots, 0)$, padded with 0 to the nearest power of two greater than $3\tau+1$. Then, we form $\overbrace{\mathbf{v}\|\mathbf{v}\|\cdots\|\mathbf{v}}^{\kappa}\|\mathbf{0}$ by packing and encrypting as many copies of $\mathbf{v}$ as possible in a single ciphertext with $\mathbf{0}$ denoting a short vector of 0's that fill out the plaintext vector. For the server, to process this ciphertext and obtain the ciphertexts required by the original protocol, we apply a mask to select one coefficient, say g_0, from each copy of $\mathbf{v}$ and then apply shifts to obtain a ciphertext encrypting $(g_0, g_0, \ldots, g_0)$.

We implemented this modification and present the results now. As expected, adding query processing on the server required us to increase the HElib capacity to 370. This reduced the security levels of the protocol to $\lambda = 126$ but did not otherwise significantly impact running time because the capacity of the processed query ciphertexts is similar to fresh ciphertexts from the original protocol. In total, there are $3\tau+1$ query coefficients to process and for our single-threaded implementation took about 72.06 seconds to complete. The client's query generation time was reduced to 1.936 seconds and communication cost is reduced to around 2.67 MB.

2) COMPARISON WITH THE PREVIOUS WORK

To determine the overhead of condition hiding, we also implemented the protocols of Kim et al. [5] for the same database. Due to keeping the same database, we cannot reduce HElib parameters since the protocol might receive a query that involves all attributes. However, we varied the number of equality conditions in the queries to the database to see how their protocol performs.

From Figure 2(a), we see that running time of the protocols scales as expected with the number of conditions. At the high

end, when every attribute is involved, the overhead needed to hide conditions is about $2-3$ seconds or roughly 10%. Below that, the overhead increases significantly since we process much fewer ciphertexts when we do not mask the attributes involved in the queries.

Remark: That said, this overhead is not necessarily inherent in the protocols' designs. Our protocols were implemented in to use a single-thread but there are opportunities to optimize with parallel computing if desired. As mentioned previously, the bulk (80%) of the protocol's execution time is on the computation of the equality conditions. Steps 2) b) and 2) a) of the SCH-/TSCH-PDQ protocols respectively are particularly amenable to parallelization.

Each $\overline{\beta}_{ij}$ can be computed independently of each other and so if we use α threads, then we can reduce the amount of time it takes to compute $\overline{\beta}_{ij}$ for each i by a factor of $\frac{\tau}{\alpha}$. This means that if there are enough resources available, the protocol's latency can be on the lower end of the graphs in Figure 2. Of course, another way to use parallelization is to use multiple threads to run multiple single-threaded executions of the protocols. Finally, the query processing step is another good target for parallelization to reduce the latency of the protocol.

VI. 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. To enhance the privacy of PDQs, we provide two PDQ protocols that hide information more than constants in a query statement to be evaluated. The first protocol supports conjunctive and disjunctive queries without revealing informations of queries to be evaluated. The second protocol extends the range of queries to be evaluated to threshold conjunctive queries. Finally, we provided implementation results of our proposed PDQ protocols.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. The part of this work was done while B. H. M. Tan was with Nanyang Technological University, Singapore.

REFERENCES

[1] Amazon. *Amazon Relational Database Service*. Accessed: Aug. 1, 2019. [Online]. Available: https://aws.amazon.com/rds/

[2] F. Olumofin and I. Goldberg, "Privacy-preserving queries over relational databases," in *Privacy Enhancing Technologies* (Lecture Notes in Computer Science), vol. 6205, M. J. Atallah and N. J. Hopper, Eds. Berlin, Germany: Springer, 2010, pp. 75–92.

[3] A. Arasu and R. Kaushik, "Oblivious query processing," in *Proc. Int. Conf. Database Theory (ICDT)*, N. Schweikardt, V. Christophides, and V. Leroy, Eds., 2014, pp. 26–37.

[4] J. H. Cheon, M. Kim, and M. Kim, "Optimized search-and-compute circuits and their application to query evaluation on encrypted data," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 1, pp. 188–199, Jan. 2016.

[5] M. Kim, H. T. Lee, S. Ling, and H. Wang, "On the efficiency of FHE-based private queries," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 2, pp. 357–363, Mar./Apr. 2018.

[6] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. ACM Symp. Oper. Syst. Princ. (SOSP)*, T. Wobber and P. Druschel, Eds., 2011, pp. 85–100.

[7] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," *Proc. VLDB Endowment*, vol. 6, no. 5, pp. 289–300, 2013.

[8] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Applied Cryptography and Network Security—ACNS* (Lecture Notes in Computer Science), vol. 7954, M. Jacobson, Jr., M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds. Berlin, Germany: Springer, 2013, pp. 102–118.

[9] T. K. Saha and T. Koshiba, "Private conjunctive query over encrypted data," in *Progress in Cryptology—AFRICACRYPT* (Lecture Notes in Computer Science), vol. 10239, M. Joye and A. Nitaj, Eds. Cham, Switzerland: Springer, 2017, pp. 149–164.

[10] T. K. Saha, Mayank, and T. Koshiba, "Efficient protocols for private database queries," in *Data and Applications Security and Privacy—DBSec* (Lecture Notes in Computer Science), vol. 10359, G. Livraga and S. Zhu, Eds. Cham, Switzerland: Springer, 2017, pp. 337–348.

[11] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 3027, C. Cachin and J. Camenisch, Eds. Berlin, Germany: Springer, 2004, pp. 1–19.

[12] L. Kissner and D. X. Song, "Privacy-preserving set operations," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 3621, V. Shoup, Ed. Berlin, Germany: Springer, 2005, pp. 241–257.

[13] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H. Zhou, "Multi-input functional encryption," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 8441, P. Q. Nguyen and E. Oswald, Eds. Berlin, Germany: Springer, 2014, pp. 578–602.

[14] 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 *Advances in Cryptology–EUROCRYPT* (Lecture Notes in Computer Science), vol. 9057, E. Oswald and M. Fischlin, Eds. Berlin, Germany: Springer, 2015, pp. 563–594.

[15] V. Pappas, M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin, "Private search in the real world," in *Proc. Annu. Comput. Secur. Appl. Conf. (ACSAC)*, R. H. Zakon, J. P. McDermott, and M. E. Locasto, Eds., 2011, pp. 83–92.

[16] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin, "Blind seer: A scalable private DBMS," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 359–374.

[17] 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 *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 395–410.

[18] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. ACM Symp. Theory Comput. (STOC)*, M. Mitzenmacher, Ed., 2009, pp. 169–178.

[20] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 6110, H. Gilbert, Ed. Berlin, Germany: Springer, 2010, pp. 24–43.

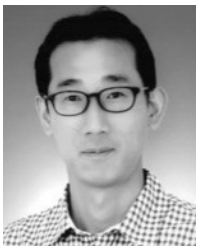
[21] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 6841, P. Rogaway, Ed. Berlin, Germany: Springer, 2011, pp. 487–504.

[22] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 9056, E. Oswald and M. Fischlin, Eds. Berlin, Germany: Springer, 2015, pp. 617–640.

[23] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proc. IEEE Symp. Found. Comput. Sci. (FOCS)*, R. Ostrovsky, Ed., Oct. 2011, pp. 97–106.

[24] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. Innov. Theor. Comput. Sci. (ITCS)*, S. Goldwasser, Ed., 2012, pp. 309–325.

[25] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig, "Improved security for a ring-based fully homomorphic encryption scheme," in *Proc. IMA Int. Conf. Cryptogr. Coding (IMACC)*, in Lecture Notes in Computer Science, vol. 8308, M. Stam, Ed. Berlin, Germany: Springer, 2013, pp. 45–64.
[26] G. S. Çetin, Y. Doröz, B. Sunar, and E. Savaş, "Depth optimized efficient homomorphic sorting," in *Progress in Cryptology—LATINCRYPT* (Lecture Notes in Computer Science), vol. 9230, K. E. Lauter and F. Rodríguez-Henríquez, Eds. Cham, Switzerland: Springer, 2015, pp. 61–80.
[27] S. Halevi and V. Shoup. (2018). *HElib: Software Library for Homomorphic Encryption*. [Online]. Available: http://github.com/shaih/HElib.git
[28] V. Shoup. (2018). *NTL: A Library for Doing Number Theory Version 11.3.2*. Accessed: Jul. 1, 2019. [Online]. Available: http://www.shoup.net/ntl/
[29] *GMP: The GNU Multiple Precision Arithmetic Library Version 6.1.2*. Accessed: Jul. 1, 2019. [Online]. Available: http://gmplib.org



MYUNGSUN KIM received the B.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 1994, the M.S. degree in computer science and engineering from the Information and Communications University (ICU), Daejeon, in 2002, and the Ph.D. degree in mathematics from Seoul National University (SNU), Seoul, in 2012. He is currently an Assistant Professor with the Department of Information Security, University of Suwon. His research interest includes multiparty computation in cryptography.

HYUNG TAE LEE received the B.S., M.S., and Ph.D. degrees in mathematics from Seoul National University, South Korea, in 2006, 2008, and 2013, respectively. He is currently an Assistant Professor with the Division of Computer Science and Engineering, College of Engineering, Jeonbuk National University, South Korea. Prior to that, he was a Research Fellow with Nanyang Technological University, Singapore. His research interests include computational number theory and cryptography.

SAN LING received the B.A. degree in mathematics from the University of Cambridge, in 1985, and the Ph.D. degree in mathematics from the University of California, Berkeley, in 1990. Since April 2005, he has been a Professor with the Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. Prior to that, he was with the Department of Mathematics, National University of Singapore. His research interests include arithmetic modular curves and applications of number theory to combinatorial design, coding theory, cryptography, and sequences.

SHU QIN REN received the bachelor's degree in computer engineering from the Hefei University of Technology, in 1997, and the Ph.D. degree in computer engineering Korea Aerospace University, in 2009. She is currently with the Security and Privacy Competence Center, Continental Automotive Singapore Pte Ltd. Her research interests include data privacy and security, secure data sharing and computing on cloud storage, distributed QoS storage, automotive security, and secure production.

BENJAMIN HONG MENG TAN received the B.Sc. and Ph.D. degrees in mathematical sciences from Nanyang Technological University, Singapore, in 2013 and 2019, respectively. He is currently a Researcher with the Institute of Infocomm Research, A*STAR, Singapore. His research interests include cryptography, multiparty computation, and secure cloud computing.

HUAXIONG WANG received the Ph.D. degree in mathematics from the University of Haifa, Israel, in 1996, and the Ph.D. degree in computer science from the University of Wollongong, Australia, in 2001. He is currently an Associate Professor with the Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. His research interests include cryptography, information security, coding theory, combinatorics, and theoretical computer science.