Optimized Search-and-Compute Circuits and Their Application to Query Evaluation on Encrypted Data

Jung Hee Cheon, Miran Kim, and Myungsun Kim

***Abstract*— Private query processing on encrypted databases allows users to obtain data from encrypted databases in such a way that the users' sensitive data will be protected from exposure. Given an encrypted database, users typically submit queries similar to the following examples: 1) How many employees in an organization make over U.S. $100 000? 2) What is the average age of factory workers suffering from leukemia? Answering the questions requires one to search and then compute over the relevant encrypted data sets in sequence. In this paper, we are interested in efficiently processing queries that require both operations to be performed on fully encrypted databases. One immediate solution is to use several special-purpose encryption schemes simultaneously; however, this approach is associated with a high computational cost for maintaining multiple encryption contexts. Another solution is to use a privacy homomorphic scheme. However, no secure solutions have been developed that satisfy the efficiency requirements. In this paper, we construct a unified framework to efficiently and privately process queries with search and compute operations. For this purpose, the first part of our work involves devising several underlying circuits as primitives for queries on encrypted data. Second, we apply two optimization techniques to improve the efficiency of these circuit primitives. One technique involves exploiting single-instruction-multiple-data (SIMD) techniques to accelerate the basic circuit operations. Unlike general SIMD approaches, our SIMD implementation can be applied even to a single basic operation. The other technique is to use a large integer ring (e.g., $\mathbb{Z}_{2^t}$) as a message space rather than a binary field. Even for an integer of k bits with $k > t$, addition can be performed using degree 1 circuits with lazy carry operations. Finally, we present various experiments performed by varying the considered parameters, such as the query type and the number of tuples.**

***Index Terms*— Encrypted databases, private query processing, homomorphic encryption.**

I. Introduction

PRIVACY homomorphism is an important concept for encrypting clear data while allowing one to perform operations on encrypted data without decryption. This concept was first introduced by Rivest *et al.* [1] and, much later, affirmed by Feigenbaum and Merritt's question [2]: Is there an encryption function $\mathsf{E}(\cdot)$ such that both $\mathsf{E}(x + y)$ and $\mathsf{E}(x \cdot y)$ can be easily computed from $\mathsf{E}(x)$ and $\mathsf{E}(y)$? Since that time, very little progress was made in determining whether such efficient and secure encryption schemes exist until 2009, when Gentry constructed such an encryption scheme [3].

Although the use of Gentry's scheme and other fully homomorphic encryption (HE) schemes (*e.g.*, [4]–[6]) theoretically allows for the secure evaluation of any function, the evaluation cost is still far from being practical for many functions. Among the various important functions, we restrict our interest to a set of functions for databases, giving rise to the following question: *Given a set of fully encrypted databases, can we construct a set of efficient functions to process queries over those encrypted databases*? If so, *is the computational cost of these functions acceptable*?

This question is the starting point of this work. To facilitate a better understanding of our approach, we would like to describe the motivation for our work from a different perspective. At present, the simplest way to search for records that satisfy a particular condition over encrypted databases is likely via *searchable encryption* (*e.g.*, [7]–[10]). However, privately processing `sum` and `avg` aggregation queries under the same conditions is performed using *homomorphic encryption* (*e.g.*, [11]–[13]). Thus, the private processing of a query that includes both matching conditions and aggregate operations requires the use of two distinct encryption techniques in parallel, *i.e.*, searchable encryption and homomorphic encryption.

Recently, Ada Popa *et al.* proposed CryptDB [14] and its extension [15], which can process general types of database queries using layers of different encryption schemes: deterministic encryption for equality condition queries, order-preserving encryption (OPE) for range queries, and HE for aggregate queries. The disadvantage of their work is that in the long run, its privacy degrades to the lowest level of data privacy provided by the weakest encryption scheme. This observation leads to a natural question: *Can we construct a solution to efficiently perform such a database query without maintaining multiple contexts of encryption*? At first glance, HE schemes appear to perfectly satisfy the requirement of processing queries on encrypted databases within a single encryption context. However, no solutions exist for *expressing* and *processing* various queries on *fully* encrypted databases in an *efficient* manner.

Manuscript received March 26, 2015; revised July 6, 2015; accepted September 22, 2015. Date of publication September 28, 2015; date of current version November 13, 2015. The work of J. H. Cheon and M. Kim was supported by Samsung Electronics Company, Ltd., under Grant 0421-20150074. The work of M. Kim was supported by the Basic Science Research Program through the National Research Foundation of Korea within the Ministry of Education under Grant 2014-R1A1A2058377. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Giuseppe Persiano. *(Corresponding author: Myungsun Kim.)*

J. H. Cheon and M. Kim are with the Department of Mathematical Sciences, Seoul National University, Seoul 151-747, Korea (e-mail: jhcheon@snu.ac.kr; alfks500@snu.ac.kr).

M. Kim is with the Department of Information Security, The University of Suwon, Hasweong 445-743, Korea (e-mail: msunkim@suwon.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2015.2483486

A. Our Results

Our main results can be summarized as follows:

- **A general framework for private query evaluation**: We provide a common platform to allow database users to work on a *single underlying cryptosystem,* to represent their queries as functions in a conceptually simple manner, and to efficiently perform these functions on fully encrypted databases.
- **Optimization of circuits and their applications to compact expressions of queries**: The foundation of our simple framework is a set of optimized circuits for the following operations: equality, greater-than comparison and integer addition. We call these operations *circuit primitives*. Our optimizations of circuit primitives are developed such that they minimize the circuit depth and the number of homomorphic operations. For this purpose, we make extensive use of single-instruction-multiple-data (SIMD) techniques to move data across plaintext slots. In general, SIMD technology allows basic operations to be performed on several data elements in parallel. In contrast, our proposal operates on packed ciphertexts of several data elements and thus enables the efficiency of the basic operations of the circuit primitives to be improved. Furthermore, we find that all circuit primitives have $\mathcal{O}(\log \mu)$ depth for μ-bit data.

  We then express more complicated queries using a combination of the optimized circuit primitives. The resulting query functions are conceptually simpler than other representations of database queries and are compact in the sense that retrieval queries require at most $\mathcal{O}(\log \mu)$ depth.
- **Further enhancement of the performance of query evaluation**: HE schemes typically use $\mathbb{Z}_2$ as a message space, meaning that their encryption algorithm encrypts each bit of a message. Although our circuit primitives function efficiently on bit encryptions, we can achieve further improvements by adopting a large integer ring (*e.g.*, $\mathbb{Z}_{2^t}$), particularly in the case of *computing* on encrypted numerical data. Even for an integer of k bits with $k > t$, addition can be performed using degree 1 circuits by processing lazy carry operations. Although this rectification requires modification of our circuit primitives, we can preserve their optimality through SIMD operations. In other words, search-and-compute queries can be processed using only $\mathcal{O}(\log \mu)$-depth circuits.
- **Experimental support**: We perform comprehensive experiments to evaluate the performance of various queries expressed using our techniques from both a theoretical and a practical perspective.

Compared with the preliminary version [16] of this paper, this journal version now includes the following new results.

1) Additional circuit compositions to support more query functionalities. Consequently, we can show how to handle an SQL query with multiple conditions, such as equality or/and greater-than conditions.
2) Improvements in the experimental studies achieved through the use of carefully selected parameters. We first investigate how the length of the plaintext modulus affects the variation in depth resulting from homomorphic evaluations. We then demonstrate how to devise our circuit primitives based on this observation.
3) Performance results from various perspectives. The weakest element of our conference paper was the lack of heuristic evidence to support the usefulness of our techniques. In this work, we present various types of experiments on different sets of possible DB queries by not only combining fine-tuned circuits but also by varying the parameters, such as the number of plaintext slots and the bit length of the plaintext space.

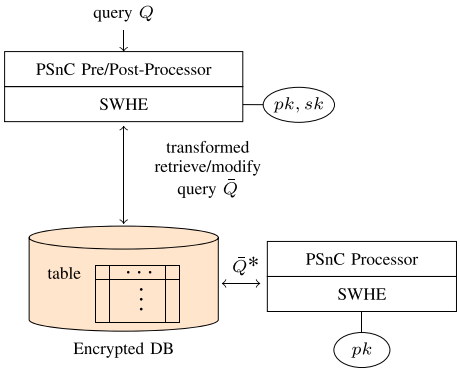


Fig. 1. Our PSnC Framework.

B. Informal Description of Our Strategy

Assuming a database consisting of N blocks, *i.e.*, $R_1 \parallel R_2 \parallel \cdots \parallel R_N$, to encrypt the record R_i, a DB user prepares a pair of public/private keys (pk, sk) for an HE scheme and publishes the public key to a DB server. The users store their encrypted records $\bar{R}_i = \mathsf{E}_{pk}(R_i)$ for $1 \le i \le N$ in the same manner as for normal write queries (*e.g.*, using the `insert − into` statement). Suppose that a user wishes to submit a retrieval query Q to the DB server. Prior to being submitted, the query Q must be properly pre-processed such that all clear messages, such as constant values, are encrypted under the public key pk. We denote this transformed query by $\bar{Q}$.

Upon receiving $\bar{Q}$, the DB server compiles it into $\bar{Q}^*$ by applying our techniques. The reader can consider a dedicated module for performing this task.[1] Hereafter, we call this module a *Private Search-and-Compute* (PSnC) processor. Next, the DB server homomorphically evaluates $\bar{Q}^*$ over the fully encrypted databases and returns the resulting ciphertexts to the user. The DB user can decrypt the output using his private key sk while receiving no additional data except for the records that satisfy the `where` conditions.

Figure 1 graphically illustrates the high-level architecture of our approach.

Organization: The remainder of this paper is structured as follows. In Section II, we briefly review the BGV-type

[1] Alternatively, one may imagine that $\bar{Q}^*$ is transformed directly by the DB user and then sent to the DB server. However, considering optimization and performance, we believe that the superior choice is for the module to be part of the DBMS.

homomorphic encryption scheme. In Section III, we construct the optimized circuits for expressing queries. Then, in Section IV, we demonstrate how to construct database queries consisting of search and/or compute operations using our circuit primitives. Section V presents our optimization techniques for further improvements in performance, and Section VI presents various experimental evaluations of our constructions. In Section VIII, we conclude with the relative coordinates of our solution in the literature, whose survey is given in Section VII, as well as several directions for future work.

II. Preliminaries

In this section, we focus on describing the efficient variant of the Brakerski-Gentry-Vaikuntanathan (BGV)-type somewhat homomorphic encryption (SWHE) scheme [6], [17] that serves as our underlying encryption scheme. We begin with a formal definition of the cryptographic assumption used in the BGV cryptosystem. In the following, we provide a description of the security model assumed in our constructions.

A. Ring-LWE (RLWE) Assumption

The ring learning with errors (RLWE) problem was first introduced by Lyubashevsky, Peikert and Regev [18]. RLWE has attracted attention because it can be stated without directly referring to lattices and also because the shortest vector problem over ideal lattices can be reduced to it. Thus, the RLWE assumption allows for several efficient SHWE schemes (*e.g.*, [5], [6]). The RLWE assumption is defined as follows.

Definition 1 (RLWE): For a security parameter κ, let $\Phi_m(X)$ be the m-th cyclotomic polynomial for an integer $m = m(\kappa)$. Let $\mathbb{A} = \mathbb{Z}[X]/\langle\Phi_m(X)\rangle$ and $\mathbb{A}_q := \mathbb{A}/q\mathbb{A}$ for an integer $q = q(\kappa) \geq 2$. Let $\chi = \chi(\kappa)$ be a distribution over $\mathbb{A}$. The $\mathrm{RLWE}_{m,q,\chi}$ problem is to distinguish two samples from polynomially many samples as follows: In the first distribution, one samples (a_i, b_i) uniformly from $(\mathbb{A}_q)^2$. In the second distribution, one first uniformly chooses s from $\mathbb{A}_q$ and then outputs $(a_i, b_i) \in (\mathbb{A}_q)^2$ by sampling $a_i \in \mathbb{R}_q$ uniformly, sampling $e_i \in \chi$ according to the distribution, and setting $b_i = a_i \cdot \mathsf{s} + e_i$. The $\mathrm{RLWE}_{m,q,\chi}$ assumption states that the $\mathrm{RLWE}_{m,q,\chi}$ problem is unfeasible.

Further details about the worst-case relation to ideal lattices are provided in [18] [Theorem 4.1 in §4].

B. The BGV-Type SWHE Scheme

For a security parameter κ, we choose an $m \in \mathbb{Z}$ that defines the m-th cyclotomic polynomial $\Phi_m(X)$. For a polynomial ring $\mathbb{A} = \mathbb{Z}[X]/\langle\Phi_m(X)\rangle$, we set the message space to $\mathbb{A}_t := \mathbb{A}/t\mathbb{A}$ for some fixed $t \geq 2$ and set the ciphertext space to $\mathbb{A}_q := \mathbb{A}/q\mathbb{A}$ for an integer q. We choose a chain of moduli $q_0 < q_1 < \cdots < q_L = q$ whereby the SWHE scheme can evaluate a depth-L arithmetic circuit. The RLWE-based SWHE scheme is described below:

- $(a, b; \mathsf{s}) \leftarrow \mathsf{Kg}(1^\kappa, h, \sigma, q_L)$: The algorithm Kg chooses a weight-h secret key s and generates an RLWE instance (a, b) relative to that secret key. We set the secret key to $sk = \mathsf{s}$ and the public key to $pk = (a, b)$.
- $\mathbf{c} \leftarrow \mathsf{E}_{pk}(x)$: To encrypt a message $x \in \mathbb{A}_t$, the algorithm chooses a small polynomial v and two Gaussian polynomials e_0 and e_1 (with variance σ^2). It outputs the ciphertext $\mathbf{c} = (c_0, c_1)$ by computing
$$(c_0, c_1) = (x, 0) + (bv + te_0, av + te_1) \bmod q_L.$$
- $x \leftarrow \mathsf{D}_{sk}(\mathbf{c})$: Given a ciphertext $\mathbf{c} = (c_0, c_1)$ at level l, the algorithm outputs $x = [c_0 - \mathsf{s} \cdot c_1]_{q_l} \bmod t$.
- $\mathbf{c}_f \leftarrow \mathsf{Ev}_{ek}(f; \mathbf{c}, \mathbf{c}')$: If the function f is an addition over ciphertexts, then the algorithm outputs the ciphertext obtained through simple component-wise addition of the two ciphertexts. If f is a multiplication over ciphertexts, then it outputs that obtained through a tensor product.

Note that our description only provides a high-level overview of the BGV cryptosystem because we intentionally omitted many details of the encryption scheme. We therefore refer the reader to [6].

C. SIMD Technique

In general, FHE (and SWHE) schemes encrypt small plaintexts (*e.g.*, $\mathbb{Z}_2$) into large ciphertexts (*e.g.*, $\mathbb{Z}_q$ with $q \gg 2$). Thus, provided that a ciphertext is able to contain a number of independent plaintexts, we can use the available memory space far more efficiently. Smart and Vercauteren [19] first noted that choosing appropriate parameters in certain FHE schemes enables those FHE schemes to support SIMD operations on finite fields of characteristic two. Their key observation was that the plaintext space $\mathbb{A}_2$ can be regarded as a vector of plaintext slots by the polynomial Chinese remainder theorem (CRT). Thus, addition and multiplication in $\mathbb{A}_2$ are performed in the same manner as the component-wise addition and multiplication of a vector of slots. In particular, because there is no need for the values in the slots to be only bits, we can use them to represent elements in $\mathbb{Z}_t$. We recommend that the reader review the original reference [20] for further details.

D. Treat Model

We will consider the following threat model. First, we assume that an SQL server is semi-honest. Thus, it should follow all specifications of our scheme. However, an adversary is allowed to access all databases maintained by a corrupted SQL server. Moreover, a corrupted DBA may become such an attacker. It is fairly plausible for an attacker to legally log into the SQL server, make an illegal copy of interesting data, and hand it over to any malicious buyer. Therefore, the DB server should learn nothing about a query beyond what is explicitly revealed (*e.g.*, the number of tuples).

Second, we assume that a DB user is also semi-honest but is not allowed to collude with an SQL server. Some corrupted DB users may create illegal copies of sensitive data; however, the volume of illegally copied data leaked at any given time is assumed to be negligible. The DB user should not be given access to data that are not part of the query result.

To formulate our security model, we follow Boneh *et al.*'s security definition [21]. Specifically, a dishonest DB server

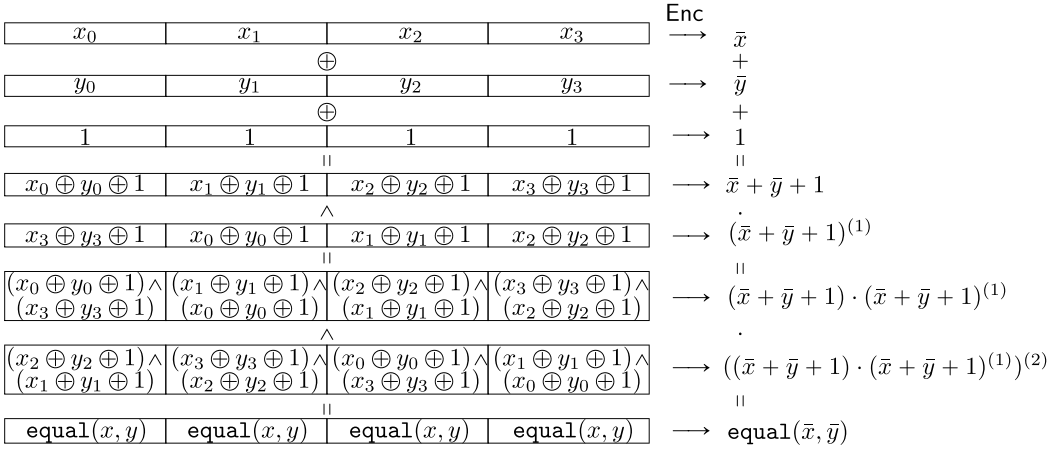


Fig. 2. An Illustration of SIMD Execution for the `equal` Circuit.

should not be able to distinguish between $\bar{Q}_0$ and $\bar{Q}_1$, where the two transformed queries $\bar{Q}_0$ and $\bar{Q}_1$ have the same syntactical form. Moreover, an adversarial DB user should not be able to distinguish between two encrypted DBs, $\overline{\mathsf{DB}}_0$ and $\overline{\mathsf{DB}}_1$, for every fixed query Q and for all pairs of DBs $(\mathsf{DB}_0, \mathsf{DB}_1)$ such that $Q(\overline{\mathsf{DB}}_0) = Q(\overline{\mathsf{DB}}_1)$.

III. Circuit Primitives

We devise three primitives: an equality circuit, a comparison circuit and an integer addition circuit. The first two circuits will be used to express the `where` clauses in SQL statements, and the last circuit will be used for the `select` clauses. We focus on a method for optimizing these circuits with respect to their depth and required homomorphic operations. For this purpose, we use SIMD along with automorphism operations.

Notation: When input messages are decomposed and encrypted in a bitwise manner, the encryption $\bar{x}$ of a message $x = x_{\mu-1} \cdots x_1 x_0$ means $\{\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_{\mu-1}\}$, where $x_i \in \{0, 1\}$. We use "+" to denote homomorphic addition and A to denote the number of additions. Similarly, for homomorphic multiplication, we use "$\cdot$" and M.

A. Equality Circuit

For two μ-bit integers x and y, we define an arithmetic circuit for the equality test as follows:

$$\texttt{equal}(\bar{x}, \bar{y}) = \textstyle\prod_{i=0}^{\mu-1} (1 + \bar{x}_i + \bar{y}_i). \tag{1}$$

The output of $\texttt{equal}(\cdot, \cdot)$ is $\bar{1}$ in the case of equality and $\bar{0}$ otherwise. In the bit-sliced implementation, we assume that one ciphertext is used per bit; therefore, we have a total of 2μ ciphertexts to evaluate the equality test. If, rather than performing regular multiplication, we multiply each term after forming a binary-tree structure, the depth of the `equal` circuit becomes $\log \mu$. Specifically, the algorithm requires two homomorphic additions to compute $1 + \bar{x}_i + \bar{y}_i$ and requires that μ ciphertexts be multiplied by each other while consuming $\log \mu$ depth.

TABLE I
Complexity of Circuit Primitives

	Circuit	Complexity
Depth	`equal`	$\log \mu$
	`comp`	$1 + \log \mu$
	`fadd`	$1 + \log(\nu - 2)$
Comp.†	`equal`	$2\mathsf{A} + (\log \mu)\,\mathsf{M}$
	`comp`	$(\mu + 1 + \log \mu)\,\mathsf{A} + (2\mu - 2)\,\mathsf{M}$
	`fadd`	$\nu\mathsf{A} + (3\nu - 5)\,\mathsf{M}$

†Comp.: Computational complexity during homomorphic evaluations

1) Our Optimizations: Our optimizations are focused on minimizing the number of homomorphic operations performed, particularly homomorphic multiplications. As shown by Smart and Vercauteren [19], we can pack each bit x_i into a single ciphertext. Next, we expand the right-hand side of Equation (1) and rearrange each term for convenience in SIMD execution. Then, we repeatedly apply SIMD operations to a vector of SIMD words. This procedure is the key to reducing the number of homomorphic multiplications from $\mu - 1$ to $\log \mu$. We provide a better description of the complexity in Table I.

For example, we consider the `equal` circuit for $\mu = 4$. We can pack each bit x_i and y_i into a corresponding single ciphertext. The application of the SIMD operations on this circuit is described in Fig. 2, where we denote the XOR and AND gate by $\oplus$ and $\wedge$. The left side represents the execution over plaintext slots and the right side represents the execution over the corresponding ciphertexts. We denote by $\bar{z}^{(i)}$ the vector obtained by applying rotation by i to each element in $\bar{z}$. Namely, we implement a single automorphism $X \mapsto X^{g^i}$ for some element $g \in \mathbb{Z}_m^*$ of order μ in the original group $\mathbb{Z}_m^*$ and the quotient group $\mathbb{Z}_m^*/\langle 2 \rangle$. In Fig. 2, a rotation operation $X \mapsto X^g$ is applied to the encryption $\bar{x} + \bar{y} + 1$ for some element $g \in \mathbb{Z}_m^*$ of order $\mu = 4$. Following the first SIMD homomorphic multiplication, we can obtain the encryption, which is entirely in the form of multiplications of

two $(x_i + y_i + 1)$s. In general, the parallel SIMD computation allows the equality circuit to be evaluated using an SWHE of depth $\log \mu$ in $\log \mu$ homomorphic multiplications.

B. Greater-Than Comparison Circuit

For two unsigned μ-bit integers, the circuit $\mathtt{comp}(\bar{x}, \bar{y})$ outputs $\bar{0}$ if $x \geq y$ and $\bar{1}$ otherwise. This operation can be recursively defined as follows:

$$\mathtt{comp}(\bar{x}, \bar{y}) = \bar{c}_{\mu-1}, \tag{2}$$

where $\bar{c}_i = (1 + \bar{x}_i) \cdot \bar{y}_i + (1 + \bar{x}_i + \bar{y}_i) \cdot \bar{c}_{i-1}$ for $i \geq 1$ with an initial value $\bar{c}_0 = (1 + \bar{x}_0) \cdot \bar{y}_0$.

1) Our Optimizations: As the first step of optimization, we express Equation (2) in the following closed form:

$$\bar{c}_{\mu-1} = (1 + \bar{x}_{\mu-1}) \cdot \bar{y}_{\mu-1} + \textstyle\sum_{i=0}^{\mu-2} (1 + \bar{x}_i) \cdot \bar{y}_i \cdot d_{i+1} d_{i+2} \cdots d_{\mu-1},$$

where $d_j = (1 + \bar{x}_j + \bar{y}_j)$. Because it has degree $\mu + 1$, we can deduce that the depth of the circuit is $\log(\mu + 1)$. Then, it is easy to see that a naïve construction of the circuit incurs $\mathcal{O}(\mu^2)$ homomorphic multiplications.

The key observation is that the closed form is expressed as a sum of products of $(1 + \bar{x}_i) \cdot \bar{y}_i$ and $(1 + \bar{x}_i + \bar{y}_i)$ terms for $i \in [0, \mu - 1]$. We are able to compute $(1 + \bar{x}_i) \cdot \bar{y}_i$ for all i using only 1 homomorphic multiplication by applying the SIMD technique. Now, we must compute $\prod_{k=i}^{\mu-1} d_k$ for each $i \in [1, \mu - 2]$. As stated above, a naïve method incurs $\mathcal{O}(\mu^2)$; however, using SIMD operations requires one to perform only $2\mu - 4$ homomorphic multiplications, thereby consuming $\log \mu$ depth. Finally, we need to multiply $(1+\bar{x}_i)\cdot\bar{y}_i$ by the result of the above computation, which also incurs only 1 homomorphic multiplication. Thus, the total number of homomorphic multiplications is equal to $2\mu - 2$.

Remark 1: We can address signed *numbers by slightly modifying the circuit. Assume that we place a sign bit in the leftmost position of a value (e.g., 0 for a positive number and 1 for a negative number) and use the two's-complement system. Then, for two μ-bit values x and y, $\mathtt{comp}(\bar{x}, \bar{y}) = \bar{c}_{\mu-1} + \bar{x}_{\mu-1} + \bar{y}_{\mu-1}$. It is evident that the case of two positive numbers corresponds to $\bar{x}_{\mu-1} = \bar{y}_{\mu-1} = \bar{0}$.*

C. Integer Addition Circuit

Suppose that for two μ-bit integers x and y and for an integer $\nu > \mu$, we construct two ν-bit integers by padding with zeros on the left. Then, a size-ν full adder $\mathtt{fadd}_\nu$ is recursively defined as follows: $\mathtt{fadd}_\nu(\bar{x}, \bar{y}) = (\bar{s}_0, \bar{s}_1, \cdots, \bar{s}_{\nu-1})$, with the sums $\bar{s}_i = \bar{x}_i + \bar{y}_i + \bar{c}_{i-1}$ and the carry-outs $\bar{c}_i = (\bar{x}_i \cdot \bar{y}_i) + ((\bar{x}_i + \bar{y}_i) \cdot \bar{c}_{i-1})$ for $i \in [1, \nu - 1]$ with initial values $\bar{s}_0 = \bar{x}_0 + \bar{y}_0$ and $\bar{c}_0 = \bar{x}_0 \cdot \bar{y}_0$. The primary reason for considering such a large full adder is to cover SQL aggregate functions with many additions.

1) Our Optimizations: Our strategy for optimization is the same as above. Namely, we express each sum and carry in closed form and minimize the number of homomorphic operations using SIMD operations. Consequently, the $\bar{s}_i$s are written as follows: $\bar{s}_i = \bar{x}_i + \bar{y}_i + \sum_{j=0}^{i-1} t_{ij}$, where $t_{ij} = \left(\bar{x}_j \cdot \bar{y}_j\right) \prod_{j+1 \leq k \leq i-1} (\bar{x}_k + \bar{y}_k)$ for $j < i - 1$ and $t_{i,i-1} = \bar{x}_{i-1} \cdot \bar{y}_{i-1}$. When $i = \nu - 1$ and $j = 0$, because $\nu - 2$ homomorphic multiplications are required, the circuit has $\log(\nu - 2)$ depth. However, we must perform an additional multiplication by $\bar{x}_j \cdot \bar{y}_j$. Thus, the total depth amounts to $\log(\nu - 2) + 1$. As before, the use of SIMD and parallelism via automorphism allows us to evaluate the integer addition circuit using only $3\nu - 5$ homomorphic multiplications, whereas a naïve method requires $\frac{(\nu^3 - 3\nu^2 + 8\nu)}{6}$ homomorphic multiplications.

IV. Search-and-Compute on Encrypted Datasets

In this section, we demonstrate how to efficiently perform queries on encrypted data using the proposed circuit primitives. We first describe our techniques in a general setting and then show how our ideas apply to database applications.

A. General-Purpose Search-and-Compute

We begin by describing our basic idea for performing a *search* operation over encrypted data. We assume that a collection of data is partitioned into N μ-bit items denoted by $R_1 \parallel \cdots \parallel R_N$ and that the data have been encrypted and stored in the form of $\bar{R}_1 \parallel \cdots \parallel \bar{R}_N$.

For a predicate φ on a ciphertext $\mathcal{C}$, a search on encrypted data outputs $\bar{R}_i$ if $\varphi(\bar{R}_i) = \bar{1}$ and $\bar{0}$ otherwise. More formally, let $\varphi : \mathcal{C} \to \{\bar{0}, \bar{1}\}$ be a predicate on encrypted data. Then, we say that $S_\varphi : \mathcal{C}^N \to \mathcal{C}^N$ is a search on the encrypted data and is defined as follows:

$$S_\varphi(\bar{R}_1, \ldots, \bar{R}_N) := (\varphi(\bar{R}_1) \cdot \bar{R}_1, \ldots, \varphi(\bar{R}_N) \cdot \bar{R}_N).$$

We then extend this operation to a more general operation on encrypted data, *i.e.*, a search-and-compute operation on encrypted data, as follows. Let $F : \mathcal{C}^N \to \mathcal{C}$ be an arithmetic function on encryptions. Then, for a restricted search $S_\varphi : \mathcal{C}^N \to \mathcal{C}^N$, we say that $(F \circ S_\varphi)(\bar{R}_1, \ldots, \bar{R}_N)$ is a search-and-compute operation on encryptions.

Furthermore, we quantify the efficiency of such search-and-compute operations on encrypted data in Theorem 1. This theorem states that if we can perform a search on encrypted data restricted by a φ that specifies only the equality operator, then a search query on the encrypted data requires $N(2\mathsf{A} + \log \mu \mathsf{M})$ homomorphic operations in total. If the predicate φ allows one to specify all the comparison operators in the set $\{<, \leq, >, \geq, \neq\}$, then we can perform $S_\varphi(\bar{R}_1, \ldots, \bar{R}_N)$ using $\mathcal{O}(\mu N)$ homomorphic multiplications.

Theorem 1: Let $\mathsf{M}(\varphi)$ and $\mathsf{M}(F)$ be the total numbers of homomorphic multiplications for φ and F, respectively. Then, we can perform $(F \circ S_\varphi)(\bar{R}_1, \ldots, \bar{R}_N)$ using $\mathcal{O}(N(\mathsf{M}(\varphi)) + \mathsf{M}(F))$ homomorphic operations. Specifically, we can perform a search on encrypted data restricted by φ using at most $\mathcal{O}(N(\mathsf{M}(\varphi)))$ homomorphic operations.

Proof: Because homomorphic multiplication dominates the performance of the operation, we may consider only operations of this type. Because the predicate φ requires $\mathcal{O}(\mathsf{M}(\varphi))$ homomorphic operations, we see that S_φ requires $\mathcal{O}(N(\mathsf{M}(\varphi)))$ homomorphic operations to compute the predicate N times. Thus, the operation uses

TABLE II
COMPLEXITY OF SEARCH QUERIES

	Query	Complexity
Depth	$(\bar{\mathsf{Q}}^*.1)$	$1+\log\mu$
	$(\bar{\mathsf{Q}}^*.2)$	$1+\log\mu+\log\tau$
Comp.	$(\bar{\mathsf{Q}}^*.1)$	$2N\mathsf{A}+N\,(1+\log\mu)\,\mathsf{M}$
	$(\bar{\mathsf{Q}}^*.2)$	$2\tau N\mathsf{A}+\tau N\,(1+\log\mu)\,\mathsf{M}$

$\mathcal{O}(\mathsf{M}(F))$ homomorphic operations to evaluate an arithmetic function F on encrypted data. Therefore, we can conclude that the total computation complexity of the search-and-compute operation on encryptions is $\mathcal{O}(N(\mathsf{M}(\varphi))+\mathsf{M}(F))$. In particular, when we consider a search on encrypted data, F can be regarded as the identity map. Therefore, we can perform a search on encrypted data restricted by φ using at most $\mathcal{O}(N(\mathsf{M}(\varphi)))$ homomorphic operations. □

B. Security Evaluation

Secrecy against a semi-honest DB server is ensured because encrypted data cannot be leaked due to the semantic security of our underlying SWHE scheme. Secrecy against a semi-honest DB user therefore follows because the result of a query expressed by our circuit primitives is equivalent to $\bar{0}$ if the specified conditions do not hold; therefore, the resulting ciphertext is equal to $\bar{0}$. This implies that the evaluated ciphertexts do not leak any information except for the number of unsatisfied tuples.

C. Applications to Encrypted Databases

We use $\mathsf{R}(A_1,\ldots,A_d)$ to denote a relation schema R of degree d consisting of attributes $A_1,\ldots,A_d$, and we use $\bar{A}_j$ to denote the corresponding encrypted attribute. As mentioned above, we use $A_j^{(i)}$ to denote the j-th attribute value of the i-th tuple, and for convenience, we assume that each has a length of μ bits.

1) Search Queries

a) Simple selection queries: Consider a simple retrieval query, as follows:

$$\texttt{select}\ A_{j_1},\ldots,A_{j_s}\ \texttt{from}\ \mathsf{R}\ \texttt{where}\ A_{j_0}=\alpha; \quad (\mathsf{Q}.1)$$

where α is a constant value.

An efficient construction of (Q.1) using our `equal` circuit is as follows:

$$\texttt{equal}\left(\bar{A}_{j_0}^{(i)},\bar{\alpha}\right)\cdot\left(\bar{A}_{j_1}^{(i)},\ldots,\bar{A}_{j_s}^{(i)}\right) \quad (\bar{\mathsf{Q}}^*.1)$$

for each $i\in[1,N]$. It follows from Theorem 1 that $(\bar{\mathsf{Q}}^*.1)$ has the complexity evaluation given in Table II.

b) Conjunctive & disjunctive queries: The query (Q.1) can be extended by adding one or more conjunctive or disjunctive conditions to the `where` clause. Consider a conjunctive query as follows:

$$\begin{aligned}&\texttt{select}\ \ A_{j_1},\ldots,A_{j_s}\\&\texttt{from}\ \ \mathsf{R}\\&\texttt{where}\ \ A_{j'_1}=\alpha_1\ \texttt{and}\ \cdots\ \texttt{and}\ A_{j'_\tau}=\alpha_\tau;\end{aligned} \quad (\mathsf{Q}.2)$$

The query (Q.2) is expressed as follows: For each $i\in[1,N]$,

$$\prod_{k=1}^{\tau}\texttt{equal}\left(\bar{A}_{j'_k}^{(i)},\bar{\alpha}_k\right)\cdot\left(\bar{A}_{j_1}^{(i)},\ldots,\bar{A}_{j_s}^{(i)}\right). \quad (\bar{\mathsf{Q}}^*.2)$$

A disjunctive query whose logical connectives are all ors can also be evaluated by changing the predicate to

$$\left(1+\prod_{k=1}^{\tau}\left(\texttt{equal}\left(\bar{A}_{j'_k}^{(i)},\bar{\alpha}_k\right)+1\right)\right).$$

With τ denoting the number of connectives, $(\bar{\mathsf{Q}}^*.2)$ requires an additional depth of $\log\tau$ compared with $(\bar{\mathsf{Q}}^*.1)$ to compute the multiplications among the τ equality tests. Table II reports the complexity analysis.

2) Search-and-Compute Queries: We continue to present important real constructions as an extension of Theorem 1, in which F is one of the built-in SQL aggregate functions: `sum`, `avg`, `count` and `max`. We begin with the case of $F=\texttt{sum}$.

a) Search-and-sum query: Consider the following `sum` query:

$$\texttt{select sum}(A_{j_1})\ \texttt{from}\ \mathsf{R}\ \texttt{where}\ A_{j_0}=\alpha; \quad (\mathsf{Q}.3)$$

As mentioned above, because our plaintext space is $\mathbb{Z}_2$, repeatedly applying simple homomorphic additions does not ensure correctness, which is the motivation for our integer addition circuit (see Section III-C). Using this circuit, we can efficiently perform (Q.3), which is expressed as follows:

$$\texttt{fadd}_{\mu+\log N}\left(\texttt{equal}\left(\bar{A}_{j_0}^{(i)},\bar{\alpha}\right)\cdot\bar{A}_{j_1}^{(i)}\right). \quad (\bar{\mathsf{Q}}^*.3)$$

Because the result of the search-and-sum query is less than $2^\mu N$, using a full adder of size $\nu=\mu+\log N$ to add all the values is sufficient. Using our optimized equality circuit, $(\bar{\mathsf{Q}}^*.3)$ requires N equality tests in total and N homomorphic multiplications for each result of the test. Thus, the total computational cost is $(2N+\nu(N-1))\mathsf{A}+(N\,(1+\log\mu)+(N-1)\,(3\nu-5)\,\mathsf{M}$ with the depth $1+\log\mu+\log N\,(1+\log(\nu-2))$ according to Theorem 2 below.

Theorem 2: Let $|\mathsf{R}|$ denote the cardinality of a set of tuples from a relation schema R. Suppose that all keyword attributes in the `where` *clause and all numerical attributes in the* `select` *clause have $\|kwd\|$ bits and $\|num\|$ bits, respectively. Then, a search-and-sum query can be processed with the depth*

$$\begin{aligned}&1+\lceil\log(\|kwd\|)\rceil\\&\quad+\lceil\log|\mathsf{R}|\rceil\cdot(1+\lceil\log(\|num\|+\lceil\log|\mathsf{R}|\rceil-2)\rceil).\end{aligned}$$

Proof: The query $(\bar{\mathsf{Q}}^*.3)$ consumes $1+\lceil\log(\|kwd\|)\rceil$ levels for the computation of all equality tests. Then, it performs $(|\mathsf{R}|-1)$ full-adder operations on the results, each of which is of size $(\|num\|+\lceil\log|\mathsf{R}|\rceil)$ and consumes $(1+\lceil\log(\|num\|+\lceil\log|\mathsf{R}|\rceil-2)\rceil)$ levels. □

b) Search-and-count query: We observe that search-and-count queries can be processed in a similar manner. For example, assume a search-and-count query with $\texttt{count}(*)$ in place of $\texttt{sum}(A_{j_1})$ in (Q.3). This query can also be efficiently processed using $\texttt{fadd}_{\log N}\left(\texttt{equal}\left(\bar{A}_{j_0}^{(i)},\bar{\alpha}\right)\right)$.

c) Search-and-avg query: To process a search-and-compute query with the `avg` aggregate function, it is sufficient to compute multiple search-and-sum queries because an average can then be obtained by applying one division after decryption.

d) Search-and-max(min) query: It is evident that one can obtain the `max` (or `min`) aggregate function by repeatedly applying the `comp` circuit primitive.

3) Join Queries: Now, we design join queries within the search-and-compute paradigm. Suppose that we have another relation $\mathsf{S}(B_1, \ldots, B_e)$ consisting of M tuples, where $M \leq N$. First, we consider a simple join query, as follows:

$$\begin{aligned}&\texttt{select } \mathsf{r}.A_{j_1}, \ldots, \mathsf{r}.A_{j_s}, \mathsf{s}.B_{j'_1}, \ldots, \mathsf{s}.B_{j'_{s'}}\\ &\texttt{from } \mathsf{R} \texttt{ as } \mathsf{r}, \mathsf{S} \texttt{ as } \mathsf{s}\\ &\texttt{where } \mathsf{r}.A_{j_k} = \mathsf{s}.B_{j'_{k'}};\end{aligned} \tag{Q.4}$$

Then, this type of query can be expressed as follows: For each $i \in [1, N], i' \in [1, M]$,

$$\texttt{equal}\left(\mathsf{r}.\bar{A}^{(i)}_{j_k}, \mathsf{s}.\bar{B}^{(i')}_{j'_{k'}}\right) \cdot \left(\mathsf{r}.\bar{A}^{(i)}_{j_1}, \mathsf{s}.\bar{B}^{(i')}_{j'_1}, \ldots\right). \tag{$\bar{\mathsf{Q}}^*$.4}$$

For fixed i and i', we suppose that each numeric-type attribute is packed in only one ciphertext. Then, the only difference from ($\bar{\mathsf{Q}}^*$.1) is that ($\bar{\mathsf{Q}}^*$.4) requires two homomorphic multiplications by the results of the search operations; thus, we must perform NM equality tests in total. Hence, the depth of the circuit needed to process ($\bar{\mathsf{Q}}^*$.4) is $1 + \log \mu$, and the computational complexity is $(2NM)\,\mathsf{A} + NM\,(2 + \log \mu)\,\mathsf{M}$.

Next, we consider an advanced join query (Q.5) with two aggregate functions $\texttt{sum}(\mathsf{r}.A_j)$ and $\texttt{count}(*)$ and the same simple condition as for (Q.4). Assuming $\texttt{sum}(\mathsf{r}.A_j) < 2^{\mu}NM$, we use a full adder of size $\nu = \mu + \log(NM)$. By contrast, the result of $\texttt{count}(*)$ is $< NM$, and it is sufficient to use a full adder of size $\log(NM)$. Thus, one candidate circuit construction for (Q.5) is as follows:

$$\begin{aligned}&\texttt{fadd}_{\mu+\log NM}\left(\texttt{equal}\left(\mathsf{r}.\bar{A}^{(i)}_{j_k}, \mathsf{s}.\bar{B}^{(i')}_{j'_{k'}}\right) \cdot \mathsf{r}.\bar{A}^{(i)}_{j}\right),\\ &\quad\texttt{fadd}_{\log NM}\left(\texttt{equal}\left(\mathsf{r}.\bar{A}^{(i)}_{j_k}, \mathsf{s}.\bar{B}^{(i')}_{j'_{k'}}\right)\right).\end{aligned} \tag{$\bar{\mathsf{Q}}^*$.5}$$

With respect to $\texttt{sum}(\mathsf{r}.A_j)$, this is identical to ($\bar{\mathsf{Q}}^*$.3) except for the number of operands of the additions. Therefore, the depth for evaluation amounts to

$$1 + \log \mu + \log(NM)\,(1 + \log(\nu - 2)),$$

and the computation complexity is

$$\begin{aligned}&(2NM + \nu(NM - 1))\mathsf{A}\\ &\quad + (NM\,(1 + \log \mu) + (NM - 1)\,(3\nu - 5))\,\mathsf{M}.\end{aligned}$$

V. Performance Improvements

There is still room to further improve the performance of the circuit primitives in Section III. Our strategies consist of three interrelated components: switching the message space $\mathbb{Z}_2$ to $\mathbb{Z}_t$; adapting the circuit primitives to $\mathbb{Z}_t$; and fine-tuning the circuit primitives, again using SIMD operations.

TABLE III
Running-Time Comparisons in $\mathbb{Z}_2$ and $\mathbb{Z}_{2^{14}}$

Msg Space	`equal` (10 bits)	`comp` (10 bits)	`add` (30 bits)
$\mathbb{Z}_2$	2.2621 ms	8.5906 ms	228.5180 ms
$\mathbb{Z}_{2^{14}}$	208.6543 ms	307.5200 ms	0.0004 ms

A. Larger Message Spaces With Lazy Carry Processing

If we encrypt messages in a bit-by-bit manner, the primary advantage is that the two comparison operations are very cheap; however, applying an integer addition circuit to encrypted data is expensive (see Table III). Instead, it would be of substantial benefit to take the message domain to be a large integer ring if doing so would allow one to efficiently evaluate the addition circuit with much lesser depth. One of the important motivations for using such a large message space is that the bit length of the keyword attributes (*e.g.*, ≤ 20 bits) in the `where` clause is generally smaller than that of the numeric-type attributes (*e.g.*, ≥ 30 bits) in the `select` clause.

Specifically, if we represent a numeric-type attribute A in the radix 2^{ω}, then we have

$$\sum_i A^{(i)} = \sum_k \sum_i [A^{(i)}]_k \cdot (2^{\omega})^k;$$

therefore, it is sufficient to compute $\sum_i [A^{(i)}]_k$ over the integers. Assuming that the plaintext modulus t is sufficiently large, we are able to perform addition without overflow in $\mathbb{Z}_t$. Note that we only need to process carry operations after computing each of them over the large integer ring.

To verify the performance improvement achieved through integer encoding, we report the running time for each circuit primitive in Table III. We suspect that integer encoding yields greater benefits in the performance of search-and-compute queries because aggregate functions rely extensively on addition. The experimental results presented in Table III used 10^2 integers in $\mathbb{Z}_t$ randomly generated by the NTL library routines.

B. Calibrating Circuit Primitives

It is clear that the use of a different message space must result in modifications to our circuit primitives. Prior to discussing our modifications in detail, we must determine certain lower bounds on the depth for homomorphic multiplication as a function of t. We have two types of homomorphic multiplications: multiplying a ciphertext by another ciphertext and multiplying a ciphertext by a known constant. We formally state the corresponding depth bounds in Theorem 3.

Theorem 3: Suppose that the native message space of the BGV cryptosystem is a polynomial ring $\mathbb{Z}_t[X]/\langle\Phi_m(X)\rangle$ and that a chain of moduli is defined by a set of primes of approximately the same size, $p_0, \cdots, p_L$, that is, the i-th modulus q_i is defined as $q_i = \prod_{k=0}^{i} p_k$. For simplicity, assume that p is the size of the p_ks. Let us denote by h the Hamming weight of the secret key. For $i \leq j$, let $\mathbf{c}$ and $\mathbf{c}'$ be normal ciphertexts at levels i and j, respectively. Then, the depth

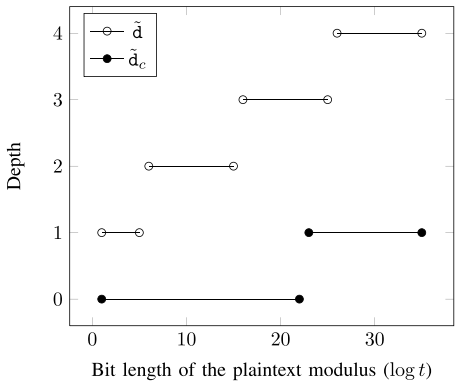


Fig. 3. Experimental Results for $\tilde{d}$ and $\tilde{d}_c$.

for the multiplication of $\mathbf{c}$ *and* $\mathbf{c}'$*, which is denoted by* $\tilde{d}$*, is the smallest nonnegative integer that satisfies the following inequality:*

$$t^2 \cdot \phi(m) \cdot (1+h) \cdot ([q_i^{-1}]_t)^2 < 6p^{2\cdot\tilde{d}}.$$

In addition, the depth for the multiplication of $\mathbf{c}$ *by a constant, which is denoted by* $\tilde{d}_c$*, is the smallest nonnegative integer for which the following inequality holds:*

$$\phi(m) \cdot (t/2)^2 < p^{2\cdot\tilde{d}_c}.$$

Proof: Before multiplying two ciphertexts, we set their noise magnitude to be smaller than the pre-set constant $B = t^2\phi(m)(1+h)/12$ via modulus switching. Subsequently, we obtain a tensor product of the ciphertexts, and the result has a noise magnitude of $2B([q_i^{-1}]_t)^2$. Next, scale-down is performed by removing small primes p_k from the current prime set of the tensored ciphertext; we use Δ to denote the product of the removed primes. We then have $2B^2([q_i^{-1}]_t)^2/\Delta^2 < B$. By assumption, it may be considered that $\Delta = p^{\tilde{d}}$, which means that $\tilde{d}$ is the smallest nonnegative integer that satisfies the inequality $2B([q_i^{-1}]_t)^2 < p^{2\cdot\tilde{d}}$.

We now consider the case in which $\mathbf{c}$ is multiplied by a constant. As above, we obtain a noise estimate of $B\cdot\phi(m)\cdot(t/2)^2$. Thus, we see that $\tilde{d}_c$ is the smallest nonnegative integer that satisfies the inequality $\phi(m)\cdot(t/2)^2 < p^{2\cdot\tilde{d}_c}$. □

In Figure 3, we graphically depict the experimental results for the depth $\tilde{d}$ of classical homomorphic multiplication and the depth $\tilde{d}_c$ of homomorphic multiplication by a constant when the bit length of the plaintext modulus ($\log t$) is varied from 1 to 35 under the following assumptions: security parameter $\kappa = 80$, the Hamming distance $h = 64$, and $m = 13981$. As a concrete example, we have $\tilde{d} = 2$ and $\tilde{d}_c = 1$ in $\mathbb{Z}_{2^{14}}$.

We now describe the basic concept that underlies our modifications. It is well known that for $x, y \in \{0,1\}$, the following properties hold:

$$x \oplus y = x + y - 2\cdot x\cdot y \quad \text{and} \quad x \wedge y = x \cdot y,$$

where $+$, $-$, and $\cdot$ are arithmetic operations over integers. Based on this observation, our equality test can be rewritten as follows:

$$\texttt{equal}(\bar{x}, \bar{y}) = \prod_{i=0}^{\mu-1} (1 - \bar{x}_i - \bar{y}_i + 2\cdot\bar{x}_i\cdot\bar{y}_i).$$

We can then see that for only a small additional cost, we can construct a new arithmetic circuit for an equality test operating on $\mathbb{Z}_t$. Next, consider the `comp` circuit on $\mathbb{Z}_t$. Recall that the closed form of $\bar{c}_{\mu-1}$ is

$$\bar{c}_{\mu-1} = (1-\bar{x}_{\mu-1})\cdot\bar{y}_{\mu-1} + \sum_{i=0}^{\mu-2}(1-\bar{x}_i)\cdot\bar{y}_i\cdot(d_{i+1}d_{i+2}\cdots d_{\mu-1}).$$

Rather than $d_j = (1+\bar{x}_j+\bar{y}_j)$, we set $d_j = (1+2\cdot\bar{x}_j\cdot\bar{y}_j - \bar{y}_j - \bar{x}_j)\cdot(1+2\cdot\bar{x}_j\cdot\bar{y}_j - 2\bar{y}_j)$. Table IV presents the complexity results for search-and-compute queries on encrypted databases of N tuples with μ-bit attributes that are obtained when using the new message space $\mathbb{Z}_t$.

TABLE IV
COMPLEXITY OF SEARCH-AND-SUM QUERIES

	Search	Complexity
Depth	`equal`	$(2+\log\mu)\,\tilde{d} + \tilde{d}_c$
	$\texttt{conj}_\tau$	$(2+\log\mu+\log\tau)\,\tilde{d} + \tilde{d}_c$
	`comp`	$(4+\log\mu)\,\tilde{d} + \tilde{d}_c$
Comp.	`equal`	$(4N-1)\,\mathsf{A} + N\,(3+\log\mu)\,\mathsf{M}$
	$\texttt{conj}_\tau$	$((3\tau+1)\,N-1)\,\mathsf{A} + \tau N\,(3+\log\mu)\,\mathsf{M}$
	`comp`	$(N\,(\mu+5+\log\mu)-1)\,\mathsf{A} + N\,(2\mu+1)\,\mathsf{M}$

VI. EXPERIMENTAL RESULTS

This section illustrates the processing performance achieved for queries expressed using our optimized circuit primitives. The essential goal of the experiments presented in this section is to verify the efficiency of our solution in terms of performance.

All of the experiments reported in our paper were performed on a machine with an Intel Xeon 2.3 GHz processor with 192 GB of main memory running the Linux 3.2.0 operating system. All methods were implemented using the GCC compiler version 4.2.1. In our experiments, we used a variant of a BGV-type SWHE scheme [17] with Shoup's NTL library [22] and Shoup-Halevi's HE library [23]. Throughout this section, when we report average running times, we exclude the computing times for data encryption and decryption.

The complete source code for our experimental implementations is available upon request from the authors.

A. Test Datasets and Queries

Our solution supports basic conjunctive and disjunctive retrieval queries with aggregate functions. Similarly, we can implement an SQL query with join conditions. Currently, because all computations are performed on ciphertexts, it is difficult to support SQL queries with the `order by` and `group by` clauses in an efficient manner. Thus, our experiments focus on retrieval and aggregate queries with

small numbers of Boolean conditions. Readers in the database community may consider that our solution can optimistically handle five—Q1, Q3, Q5, Q10, and Q15—out of 22 TPC-H queries, but without the `order by` and `group by` clauses.

For simple retrieval queries, our experiments were conducted over a table of single 300- or 320-tuples consisting of all numeric-type attributes, and all attribute values were 15-bit random integers. In the case that these queries have comparison operators rather than equality, our experiments were conducted while varying the number of tuples (10^2, 10^3, and 10^4), and each tuple had $75 \sim 200$ attributes of 14- and 17-bit lengths. For search queries with aggregate functions, our experiments used a table with 10^2, 10^3, and 10^4 tuples, and each tuple was composed of $75 \sim 200$ attributes, whose sizes were at most 16-bit. Join queries were tested over two tables whose tuple sizes were 10 and 100, respectively, but every tuple had 300 attributes in a 15-bit integer domain.

In the remaining sections, we denote by N the number of tuples, by s the number of attributes, and by τ the number of comparison operators.

B. Adjusting the Parameters

The keyword attributes are expressed in a bit-by-bit manner, and each bit is an element of $\mathbb{Z}_{2^r}$. In addition, the numeric-type attributes are expressed using the radix 2^ω but are elements of the same space, $\mathbb{Z}_{2^r}$. We begin by observing the following relation among the parameters.

Theorem 4: Let A be a numeric-type attribute. For a positive integer $\omega \geq 1$, suppose that each attribute is written as $A = \sum_k [A]_k \cdot (2^\omega)^k$ with $0 \leq [A]_k < 2^\omega$. Then, to process a search-and-sum query, one can consider a plaintext modulus with $r = \Theta(\omega + \log(\varepsilon N))$.

Proof: The purpose of the theorem is to provide a bound on the size of plaintext moduli; therefore, we simply omit the overbars for all variables. Let us use φ to denote a predicate on encrypted data and A^* to denote a keyword attribute. Then, a search-and-sum query can be written as

$$\sum_i S_\varphi(A^*, \alpha) \cdot A^{(i)} = \sum_k \left(\sum_i S_\varphi(A^*, \alpha) \cdot [A^{(i)}]_k \right) \cdot (2^\omega)^k.$$

We then have

$$\sum_i S_\varphi(A^*, \alpha) \cdot [A^{(i)}]_k < 2^\omega \sum_i S_\varphi(A^*, \alpha) = 2^\omega \cdot (\varepsilon N).$$

Thus, for a database with N records, it is sufficient to choose r such that $2^\omega \cdot (\varepsilon N) \leq 2^r$. Note that the larger we make the plaintext modulus 2^r, the more noise there is in the ciphertexts, and thus, the faster we consume the ciphertext level. Therefore, it appears that $\omega + \log(\varepsilon N)$ is a tight bound on the parameter r. □

One may wonder why $S_\varphi(\cdot, \ldots)$ does not take multiple keyword attributes in the proof. However, because we consider the selectivity ratio, it does not need to do so. In our experiments, we varied the selectivity ratio from 5 to 40% and plotted the average running times of queries over databases.

TABLE V
PERFORMANCE FOR ($\bar{\mathsf{Q}}^*$.1) AND ($\bar{\mathsf{Q}}^*$.2)

Msg Space	τ	s	m	L	Timing	Comm.
$\mathbb{Z}_{2^{15}}$	1	300	13981	14	0.025s	1.57MB
	2	300	13981	16	0.033s	1.66MB
	4	320	20485	18	0.066s	2.42MB

C. Experiments for Search Queries

We measured the running times of the simple retrieval queries ($\bar{\mathsf{Q}}^*$.1) and ($\bar{\mathsf{Q}}^*$.2), which merely require search operations over encryptions. We assumed that a numeric-type attribute was expressed using the radix 2^{15}. We took $\mathbb{Z}_{15}$ as the message space so that each attribute was encoded into two slots per ciphertext.

The query ($\bar{\mathsf{Q}}^*$.1) is efficiently processed using only the equality test. We chose a ring modulus m such that the number of plaintext slots was divisible by 10, and there existed an element $g \in \mathbb{Z}_m^*$ of order 10 in $\mathbb{Z}_m^*$ and $\mathbb{Z}_m^*/\langle 2 \rangle$. Thus, an entire keyword attribute could be packed into only one ciphertext. Furthermore, there was a Frobenius automorphism of cyclic right shifts over those 10 plaintext bits. We used $m = 13981$; thus, each of the ciphertexts held 600 plaintext slots.

The query ($\bar{\mathsf{Q}}^*$.2) has two or more equality tests in the `where` clause (*i.e.*, $\tau \geq 2$). We performed experiments for $\tau = 2$ and $\tau = 4$. For $\tau = 2$, we used $m = 13981$ as before. For the $\tau = 4$ case, we chose $m = 20485$ to support a larger number of multiplications; thus, each ciphertext held 640 plaintext slots.

For $N = 1$, the experiment for the query ($\bar{\mathsf{Q}}^*$.1) is presented in the top row of Table V and that for ($\bar{\mathsf{Q}}^*$.2) is presented in the bottom two rows of Table V, where τ is the number of comparison operaters, s is the number of numeric-type attributes, m is a ring modulus of the plaintext space, and L is the number of ciphertext moduli.[2]

D. Experiments for Search-and-Sum Queries

We conducted a series of additional experiments to measure the performance for search-and-sum queries. Because each ciphertext can hold ℓ plaintext slots of elements in $\mathbb{Z}_{2^r}$ and because a numeric-type attribute with a length of 30 bits is encoded into $\tilde{\omega}$ ($= \lceil 30/\log(2^\omega) \rceil = \lceil 30/\omega \rceil$) slots, we can process s ($= \lfloor \ell/\tilde{\omega} \rfloor$) attributes per ciphertext.

At first glance, a larger ω would appear to be better. However, if ω is too large, then by Theorem 4, the plaintext modulus 2^r becomes large, which results in an increased circuit depth. Therefore, we must choose a sufficiently large ω such that the resulting plaintext space is not too large.

We divided our experiment into four cases by predicate type: (a) single equality, (b) multiple equality, and (c) single comparison. We recommend that the reader review the original reference [24] for other experiments in greater detail.

[2]We assumed that the DB user sends his public key as well as the encryptions of database records to the DB server once and for all. Hence we considered only the number of bits of queries and corresponding responses for the communication complexity in Table V.

TABLE VI
EXPERIMENTS FOR CASE I ($\tau = 1$)

N	ε	Msg	Rdx.	s	L	Timing	Comm.
10^2	$< 16\%$	$\mathbb{Z}_{2^{14}}$	2^{10}	200	14	3.69s	1.36MB
	$< 32\%$	$\mathbb{Z}_{2^{15}}$			15	3.89s	1.46MB
10^3	$\leqslant 6\%$	$\mathbb{Z}_{2^{16}}$	2^{10}	200	15	38.78s	1.46MB
	$\leqslant 25\%$		2^8	150		51.64s	
10^4	$\leqslant 10\%$	$\mathbb{Z}_{2^{16}}$	2^6	120	15	681.05s	1.46MB
	$\leqslant 20\%$		2^5	100		817.26s	
	$\leqslant 40\%$		2^4	75		1089.68s	

TABLE VII
EXPERIMENTS FOR CASE II ($\tau = 2$)

N	ε	Msg	Rdx.	s	L	Timing		Comm.
10^2	$< 16\%$	$\mathbb{Z}_{2^{14}}$	2^{10}	200	16	4.81s	4.84s	1.44MB
	$< 32\%$	$\mathbb{Z}_{2^{15}}$			17	5.12s	5.26s	1.56MB
10^3	$\leqslant 6\%$	$\mathbb{Z}_{2^{16}}$	2^{10}	200	17	51.63s	52.14s	1.56MB
	$\leqslant 25\%$		2^8	150		68.83s	69.52s	
10^4	$\leqslant 10\%$	$\mathbb{Z}_{2^{16}}$	2^6	120	17	913.18s	926.11s	1.56MB
	$\leqslant 20\%$		2^5	100		1095.81s	1111.33s	
	$\leqslant 40\%$		2^4	75		1261.08s	1481.77s	

TABLE VIII
EXPERIMENTS FOR CASE II ($\tau = 4$)

N	ε	Msg	Rdx.	s	L	Timing		Comm.
10^2	$< 16\%$	$\mathbb{Z}_{2^{14}}$	2^{10}	213	18	9.79s	9.86s	2.12MB
	$< 32\%$	$\mathbb{Z}_{2^{15}}$			19	10.24s	10.28s	2.27MB
10^3	$\leqslant 6\%$	$\mathbb{Z}_{2^{16}}$	2^{10}	213	19	101.86s	105.15s	2.27MB
	$\leqslant 25\%$		2^8	160		135.59s	139.97s	
10^4	$\leqslant 10\%$	$\mathbb{Z}_{2^{16}}$	2^6	128	19	1788.19s	1800.84s	2.27MB
	$\leqslant 20\%$	$\mathbb{Z}_{2^{17}}$	2^6	128	20	1850.70s	1864.36s	
	$\leqslant 40\%$	$\mathbb{Z}_{2^{17}}$	2^5	106	20	2234.81s	2251.30s	

a) Case I: single equality: In this case, one equality test is contained in the `where` clause. We chose the plaintext spaces by Theorem 4 while varying N from 10^2 to 10^4. Using the same encoding method for keyword attributes as in ($\bar{\mathsf{Q}}$*.1), we also used $m = 13981$ so that we could have processed SIMD operations with 600 plaintext slots in Table VI.

b) Case II: multiple equality: In this case, we performed experiments with $\tau = 2$ and $\tau = 4$ equality tests in the `where` clause. When $\tau = 2$, we used $m = 13981$ as before, while $m = 20485$ for the other case. Compared with queries in the conjunctive form, disjunctive-form queries require more addition operations. However, both types of queries require the same depth; therefore, their running times are not significantly different.

The results are presented in Table VII and Table VIII (the 6th column of each table consists of two parts: the left part corresponds to conjunctive-form queries, and the right part corresponds to disjunctive-form queries.)

c) Case III: single comparison: In this case, one greater-than comparison is contained in the `where` clause. For the experiments, we used $m = 20485$ for the case of $L = 20$, whereas in all other experiments, we used $m = 13981$. We report the experimental results in Table IX.

TABLE IX
EXPERIMENTS FOR CASE III

N	ε	Msg	Rdx.	s	L	Timing	Comm.
10^2	$< 16\%$	$\mathbb{Z}_{2^{14}}$	2^{10}	200	17	9.98s	5.79MB
	$< 32\%$		2^9	150		13.31s	
10^3	$\leqslant 6\%$	$\mathbb{Z}_{2^{14}}$	2^8	150	17	133.12s	5.79MB
	$\leqslant 25\%$		2^6	120		166.40s	
10^4	$\leqslant 10\%$	$\mathbb{Z}_{2^{14}}$	2^4	75	17	2805.97s	5.79MB
	$\leqslant 20\%$	$\mathbb{Z}_{2^{17}}$	2^6	128	20	3116.66s	5.79MB
	$\leqslant 40\%$	$\mathbb{Z}_{2^{17}}$	2^5	106	20	3763.51s	10.66MB

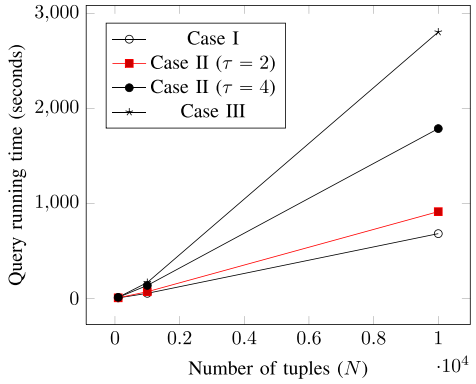


Fig. 4. Experimental Results for Search-and-Sum Queries.

TABLE X
EXPERIMENTS FOR ($\bar{\mathsf{Q}}$*.4)

$\tau = 1$ and $s = s' = 300$		
$N = M = 10$	$N = 10^2, M = 10$	$N = M = 10^2$
2.53 s	26.93 s	265.38 s

The results for Case IV are very similar to those for Case II. Thus, because of space limitations, we have omitted the experimental results for Case IV.

To facilitate comparison, in Figure 4, we graphically depict the experimental results described above for a fixed selectivity ratio ε of 10%.

E. Experiments for Join Queries

In this subsection, we report several experiments for join queries. We measured the average running time required for processing a single equality test while varying N and M from 10 to 10^2 for fixed $s = s' = 300$. Table X reports the experimental results for ($\bar{\mathsf{Q}}$*.4). As N and M increase, the running time of the algorithm also linearly increases. Table XI presents the experimental results for ($\bar{\mathsf{Q}}$*.5) with the selectivity ratio fixed at 10%. Because the experiments for ($\bar{\mathsf{Q}}$*.5) required more aggregation operations than those for ($\bar{\mathsf{Q}}$*.4), the queries required a longer time to run.

TABLE XI
EXPERIMENTS FOR ($\bar{\mathsf{Q}}^*$.5)

$\tau = 1$		
$N = M = 10$ $s = s' = 200$	$N = 10^2, M = 10$ $s = s' = 150$	$N = M = 10^2$ $s = s' = 120$
3.79 s	50.84 s	680.27 s

VII. LITERATURE REVIEW

We begin by presenting the results most closely related to our work, which are aimed at supporting general DB queries in a private manner. TrustedDB [25] has achieved the desired goal, but it requires a secure co-processor for processing on sensitive data. Hacigümüs *et al.* [26] attempted to support general DB queries in a private manner. Hore *et al.* [27] claimed that their schemes could support range queries while maintaining privacy. However, they were later found to reveal the underlying data distributions. Recently, Ada Popa *et al.* showed that their CryptDB [14] system can very efficiently process even general types of database queries. However, because CryptDB is based on layering in deterministic encryption, OPE, and *additive* HE, it needs to decrypt encrypted attributes if multiplication operations on them are required. Quite recently, Tu *et al.* [15] proposed an enhanced variant of CryptDB on the practical side, called Monomi. Their solution introduces some elaborate techniques for achieving better performance. For example, because Paillier's cryptosystem cannot support multiplications of two ciphertexts, multiplication of two column values should be encrypted in advance, and furthermore, some queries should be rewritten for being executed in a client machine. Monomi reuses Paillier ciphertexts to save the amount of space wasted in encryption. However, Monomi inherits most of the security drawbacks from CryptDB.

There have been many other results for special types of queries, such as Boolean queries or aggregate queries. For example, private information retrieval (PIR) [28] enables a DB user to retrieve a tuple from a DB without revealing which tuple the DB user is retrieving. However, the DB user must provide a set of indices of target tuples [29], and it may not have any such index information. Olumofin and Goldbeg [30] extended PIR to SQL-enabled PIR for the private processing of general DB queries. They focused on ensuring query privacy but not data privacy.

Searchable encryption (*e.g.*, [7], [9], [10]) allows a DB user to search for a specific keyword by submitting a trapdoor without revealing any keywords or original data. With this searchability, the database community has proposed various applications. Yang *et al.* [31] and Wen *et al.* [32] proposed a scheme for the private processing of conjunctive queries, and recently, Lin and Wong [33] proposed a solution for private disjunctive queries using OPE and bucketization. Boneh and Waters [34], Lu [35], and Wen *et al.* [36] showed that predicate encryption can be used in private range queries. However, schemes falling in this category may support only search functionality, and some of them based on weakly secure encryption such as OPE might leak the coupling distribution between plaintext and ciphertext domains. Other works, such as [37] and [38], assume that there is a set of mutually trusted and host participants. Ge and Zdonik considered the same security model [39]. Their scheme, however, is restricted to aggregate queries.

We note that there are solutions designed by somewhat different approaches. Pappas *et al.* [40] proposed a system for private queries by efficiently fusing together with Yao's garbled circuit and Bloom filter. Unfortunately, their solution only supports search queries. Later, Wang and Ravishankar's scheme [41] utilized scalar-product encryption, and Arasu and Kaishik [42] showed that oblivious RAM can be used in private query processing. However, the former works on a too weak security model, and the latter provides a purely theoretical result.

VIII. FURTHER DISCUSSION

CryptDB and Monomi are located on the other extreme side with high performance, while our general solution is the one extreme with high security. Here, we list a few fascinating open problems that remain.

- The grand challenge is to improve the performance of underlying FHE schemes. A more modest goal is to find a way to minimize the usage of an FHE scheme by load-balancing with group homomorphic encryption compatible with the FHE scheme. Other tools, perhaps borrowing from a different representation of queries other than circuits, may lead to more efficient performance.
- Although FHE has a long way to go in its practical uses, the good news is that there have been significant improvements in HE schemes (*e.g.*, [43]–[45]). We expect to have much faster performance in near future by applying more efficient FHE schemes to our current protocol instead of the BGV scheme.
- Our results to date scratch the surface of an FHE-based general framework for private query evaluation and only implement general but relatively simple queries. These are just starting points on FHE-based private query processing and we need to implement a prototype of our solution working on top of a real DBMS, such as MySQL, which can be a final goal of our research.

REFERENCES

[1] 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.
[2] J. Feigenbaum and M. Merritt, "Open questions, talk abstracts, and summary of discussions," in *Proc. DIMACS*, vol. 2. 1991, pp. 1–45.
[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM STOC*, 2009, pp. 169–178.
[4] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 6110, H. Gilbert, Ed. Berlin, Germany: Springer-Verlag, 2010, pp. 24–43.
[5] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proc. 52nd Annu. Symp. FOCS*, 2011, pp. 97–106.
[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. 3rd ITCS*, 2012, pp. 309–325.
[7] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.

[8] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 3027, C. Cachin and J. Camenisch, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 506–522.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.

[10] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 4622, A. Menezes, Ed. Berlin, Germany: Springer-Verlag, 2007, pp. 535–552.

[11] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 196, G. R. Blakley and D. Chaum, Eds. Berlin, Germany: Springer-Verlag, 1984, pp. 10–18.

[12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 1592, J. Stern, Ed. Berlin, Germany: Springer-Verlag, 1999, pp. 223–238.

[13] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. 2nd TCC*, vol. 3378. 2005, pp. 325–341.

[14] R. Ada Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. 23rd ACM SOSP*, 2011, pp. 85–100.

[15] 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.

[16] J. H. Cheon, M. Kim, and M. Kim, "Search-and-compute on encrypted data," in *Financial Cryptography Data Security* (Lecture Notes in Computer Science), vol. 8976, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds. Berlin, Germany: Springer-Verlag, 2015, pp. 142–159.

[17] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 7417, R. Safavi-Naini and R. Canetti, Eds. Berlin, Germany: Springer-Verlag, 2012, pp. 850–867.

[18] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 6110, H. Gilbert, Ed. Berlin, Germany: Springer-Verlag, 2010, pp. 1–23.

[19] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," Cryptol. ePrint Arch., Tech. Rep. 133, 2011.

[20] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs, Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[21] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Proc. ACNS*, vol. 7954. 2013, pp. 102–118.

[22] V. Shoup. (2009). *NTL: A Library for Doing Number Theory*. [Online]. Available: http://www.shoup.net/ntl/

[23] V. Shoup and S. Halevi, "Design and implementation of a homomorphic-encryption library," IBM, Tech. Rep., 2013.

[24] J. H. Cheon, M. Kim, and M. Kim, "Search-and-compute on encrypted data," Cryptol. ePrint Arch., Tech. Rep. 812, 2014.

[25] S. Bajaj and R. Sion, "TrustedDB: A trusted hardware-based database with privacy and data confidentiality," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 752–765, Mar. 2014.

[26] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. SIGMOD*, 2002, pp. 216–227.

[27] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proc. 13th Int. Conf. VLDB*, 2004, pp. 720–731.

[28] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[29] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, "Protecting data privacy in private information retrieval schemes," in *Proc. 13th Annu. ACM STOC*, 1998, pp. 151–160.

[30] 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-Verlag, 2010, pp. 75–92.

[31] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving queries on encrypted data," in *Proc. 11th ESORICS*, vol. 4189. 2006, pp. 479–495.

[32] M. Wen, R. Lu, J. Lei, X. Liang, H. Li, and X. Shen, "ECQ: An efficient conjunctive query scheme over encrypted multidimensional data in smart grid," in *Proc. IEEE GLOBECOM*, Dec. 2013, pp. 796–801.

[33] S.-P. Li and M.-H. Wong, "Privacy-preserving queries over outsourced data with access pattern protection," in *Proc. IEEE ICDM Workshops*, Dec. 2014, pp. 581–588.

[34] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. 4th TCC*, vol. 4392. 2007, pp. 535–554.

[35] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud," in *Proc. NDSS*, 2012, pp. 1–17.

[36] M. Wen, R. Lu, K. Zhang, J. Lei, X. Liang, and X. Shen, "PaRQ: A privacy-preserving range query scheme over encrypted metering data for smart grid," *IEEE Trans. Emerg. Topics Comput.*, vol. 1, no. 1, pp. 178–191, Jun. 2013.

[37] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin, "Secure anonymous database search," in *Proc. ACM Workshop CCSW*, 2009, pp. 115–126.

[38] S. S. M. Chow, J.-H. Lee, and L. Subramanian, "Two-party computation model for privacy-preserving queries over distributed databases," in *Proc. NDSS*, 2009, pp. 1–16.

[39] T. Ge and S. Zdonik, "Answering aggregation queries in a secure system model," in *Proc. 33rd Int. Conf. VLDB*, 2007, pp. 519–530.

[40] V. Pappas *et al.*, "Blind seer: A scalable private DBMS," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 359–374.

[41] P. Wang and C. V. Ravishankar, "Secure and efficient range queries on outsourced databases using R̂-trees," in *Proc. IEEE 29th ICDE*, Apr. 2013, pp. 314–325.

[42] A. Arasu and R. Kaushik, "Oblivious query processing," in *Proc. ICDT*, 2014, pp. 26–37.

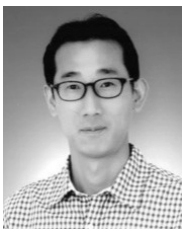
[43] J.-S. Coron, T. Lepoint, and M. Tibouchi, "Scale-invariant fully homomorphic encryption over the integers," in *Proc. 17th Int. Conf. PKC*, vol. 8383. 2014, pp. 311–328.

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

[45] S. Halevi and V. Shoup, "Bootstrapping for HElib," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 9056, E. Oswald and M. Fischlin, Eds. Berlin, Germany: Springer-Verlag, 2015, pp. 641–670.

Jung Hee Cheon received the B.S. and Ph.D. degrees in mathematics from the Korea Advanced Institute of Science and Technology, in 1991 and 1997, respectively. He was a Senior Researcher with the Electronics and Telecommunications Research Institute, and a Visiting Scientist with Brown University. He was an Assistant Professor with the Information and Communications University for two years. He is currently a Professor with the Department of Mathematical Sciences, Seoul National University, and the Director of the Cryptographic Hard Problems Research Initiatives. His research interests include computational number theory, cryptography, and information security.

Miran Kim received the B.S. degree in mathematical education and the M.S. degree in mathematics from Seoul National University, Seoul, Korea, in 2010 and 2012, respectively, where she is currently pursuing the Ph.D. degree in mathematics. Her current research interests include computational number theory, cryptography, and information security.



Myungsun Kim received the B.S. degree in computer science and engineering from Sogang University, Seoul, Korea, in 1994, the M.S. degree in computer science and engineering from Information and Communications University, Daejeon, in 2002, and the Ph.D. degree in mathematics from Seoul National University, Seoul, in 2012. He was with the Digital Media Research and Development Center, Samsung Electronics, until 2008. He is currently an Assistant Professor with the Department of Information Security, The University of Suwon. His research interests focus on multiparty computation in cryptography.