electronics

MDPI

Article

An Efficient Search Algorithm for Large Encrypted Data by Homomorphic Encryption

Pyung Kim [1,†], Eunji Jo [2,†] and Younho Lee [3,*]

[1] Advanced Software Research Center, Incheon National University, Incheon 22012, Korea; pkim84@inu.ac.kr
[2] Department of Software Design and Analysis, Seoul National University of Science and Technology, Seoul 01811, Korea; ejjo321@gmail.com
[3] ITM Programme, Department of Industrial Engineering, Seoul National University of Science and Technology, Seoul 01811, Korea
* Correspondence: younholee@seoultech.ac.kr; Tel.: +82-2-970-7283
† These authors contributed equally to this work.

Abstract: The purpose of this study is to provide an efficient search function over a large amount of encrypted data, where the bit length of each item is several tens of bits. For this purpose, we have improved the existing hybrid homomorphic encryption by enabling the longer data items to be stored while using multiple encrypted databases and by suggesting an improved search method working on top of the multiple instances of the database. Further, we found the optimal number of databases to be needed when 40-bit information, such as social security number, is stored after encryption. Through experiments, we were able to check the existence of a given (Korean) social security number of 13 decimal digits in approximately 12 s from a database that has 10 million encrypted social security numbers over a typical personal computer environment. The outcome of this research can be used to build a large-scale, practical encrypted database in order to support the search operation. In addition, it is expected to be used as a method for providing both security and practicality to the industry dealing with credit information evaluation and personal data requiring privacy.

Keywords: secure keyword search; homomorphic encryption; applied cryptography; security

Citation: Kim, P.; Jo, E.; Lee, Y. An Efficient Search Algorithm for Large Encrypted Data by Homomorphic Encryption. *Electronics* **2021**, *10*, 484. https://doi.org/10.3390/electronics10040484

Academic Editor: Riccardo Sisto

Received: 15 December 2020
Accepted: 8 February 2021
Published: 18 February 2021

1. Introduction

In recent times, it is not uncommon to say that there are many personalized services utilizing IT technology. For personalized services, the use of personal information, such as personal identification numbers, is inevitable. Unfortunately, such use has led to a sharp increase in the number of incidents involving the leakage of personal information [1]. In particular, the Korean residence registration numbers (RRNs), which have roles that are similar to social security numbers in the U.S., have been widely used as personal identification numbers for a long time; they are more extensively collected along with personal information, recorded, and managed in multiple repositories, owing to the progress of the information society. RRN has a strong personal identification value and it can be used as an identity verification tool in all fields of activity; hence, if it is exposed, it is possible not only to impersonate another person, but also use it for fraud or illegal collection, and marketing of various target advertisements. Therefore, unprotected RRN can cause serious privacy infringement [2–4].

To solve these problems, technical methods for preserving the privacy of personal sensitive information, such as RRN, are being studied along with the improvement of the system for strengthening the management and supervision of personal information holders [5]. Homomorphic encryption (HE) is one of the methods considered [6–17].

The hybrid HE [18,19] technique, which provides an efficient matching search, has recently been proposed as an HE method. It aims to quickly search for the presence or absence of a specific sequence from a set of encoded sequence information. We improve

this method and propose an efficient search method on encrypted RRN database. The problem with the existing hybrid HE method is that the size of the data item to be searched is limited to 11 bits; hence, there is not enough space to store RRNs that require more than 40 bits each. In addition, the existing hybrid HE methods target approximately tens of thousands of data for searching. Thus, it seems to be infeasible to use these methods for real-time processing of approximately 10 million data points.

In this paper, we focus on this problem and propose a new method that provides a search function for ten million encrypted data in practical time. The proposed method is based on the existing hybrid HE method [18,19]. However, by extending the maximum bit length of plaintexts that can be encrypted, it is possible to search for longer data than that using the previous method in a short time. In the basic database constituting the existing hybrid HE method [18], the size of the search keyword and data item to be searched is limited to 11 bits each, and a method of increasing the keyword to 22 bits using two databases is proposed. We increase the storage efficiency by expanding the database not only the size of the search keyword, but also the size of the data item to be searched to 22 bits. We also propose a method that uses three or more multiple encrypted databases and enables a parallel search on them. Therefore, the search performance of the proposed method can be improved. In addition, when a keyword search is performed on the part of the RRNs, it is necessary to process data items that share the same keyword, unlike the existing hybrid HE method. Therefore, the proposed method also handles it together in the process of expanding the database.

We prove the efficacy of the proposed method by randomly generating tens of millions of RRNs and then measuring the elapsed times for encryption, database creation, and search. In a conventional personal computer (PC) environment, the proposed method takes less than 10 s to search for one of the 10 M data. This result is approximately 43.9 times faster than the Paillier encryption-based search method [9], which, as far as we know, is one of the most widely used efficient HE scheme.

In addition, it is known that the Ring LWE problem [20], which is the security assumption of the encryption system used in the hybrid HE method [18,19] that is the basis of the proposed method, is safe from quantum computing attacks. Therefore, the framework that is described in this paper is expected to be robust against quantum computer attacks in the future.

The remainder of this paper is organized, as follows. Section 2 describes the preliminaries and related work. Section 3 provides the proposed method for efficient search of large-scale encrypted data. Section 4 describes the results of the performance evaluation. Section 5 presents a complementary discussion on the issues that are related to the implementation of the proposed work. Finally, Section 6 presents the conclusions, limitations, and directions for future research.

2. Preliminaries and Related Work

This section discusses the preliminaries that help to understand this work and provide related work. In the first subsection, we present the basic concept of HE and explain somewhat HE (SWHE), where the hybrid HE can be classified. Subsequently, we introduce the hybrid HE. The second subsection covers related work. Finally, the third subsection briefly describes the structure of the RRNs to be stored in the database in this study.

2.1. Preliminaries

Homomorphic Encryption (HE): HE is an encryption technique that enables computation between the plaintexts hidden in their corresponding ciphertexts. The result of the computation in HE is another ciphertext that contains the correct computation result between plaintexts. In conventional encryption algorithms, once a plaintext is encrypted, the hidden plaintext in the ciphertext cannot be manipulated, because the ciphertext completely hides the plaintext, and nothing is supported by the encryption algorithms, except the decryption and encryption operations. Therefore, the ciphertexts must be decrypted to extract the plaintexts to perform operations on the plaintexts embedded in ciphertexts. In

addition, to create a ciphertext containing the operation result, the plaintext of the operation result must be encrypted again. However, HE can perform operations without decrypting the text and obtain the ciphertext containing the result of the computation. Rivest and Adelman first proposed the concept of HE, who were the proponents of the RSA encryption method, named after them [10]. They [10] proposed "privacy homomorphism" as a way to obtain sensitive information from company-owned customers and suggested five encryption methods. However, these five schemes have security problems. Almost all of the proposed HEs until the mid-1990s have been shown to have security problems [11–14]. The concept of "operation in the encrypted state", as presented in [10], however, became the basic idea of HE and, since then, some secure HE algorithms have been proposed [9,15–17]. Unfortunately, they support limited functionality [15]: to be specific, some algorithms support only 1 bit XOR operation [16], some support only multiplication, whereas others [9,17] support only addition on the underlying plaintexts hidden in the ciphertexts.

Somewhat HE (SWHE): SWHE [7,8,21] refers to homomorphic encryption methods that support a limited number of consecutive multiplications operations from a fresh ciphertext, which is, the ciphertext as a result of encryption, while the number of consecutive addition operations allowed is practically infinite. In generating the security parameters, the number of consecutive maximum multiplications operations that can be performed for a ciphertext, which is, the multiplicative depth of the circuit to be performed, is determined. The SWHE scheme is defined by the following algorithms in Figure 1.

$ParamGen(1^{\lambda}) \rightarrow params$: a parameter generation algorithm that takes the security parameter and outputs a parameter that is used for key generation.

$KeyGen(params) \rightarrow (pk, evk, sk)$: a key generation algorithm that takes the parameter *params* and outputs the public key *pk* for encryption, the evaluation key *evk* for computation over encrypted data, and the secret key *sk* for decryption.

$Enc_{pk}(m) \rightarrow c$: an encryption algorithm that takes a plaintext and a public key and outputs a ciphertext that contains the plaintext. The ciphertext maintains the maximum number of multiplication operations that can be done with it. Whenever a multiplication is performed with it, the resultant ciphertext has the maximum possible multiplication number that is decremented by one from that of the input ciphertext.

$Dec_{sk}(c) \rightarrow m$: *A* decryption algorithm that takes a ciphertext and a secret key and outputs a plaintext in the ciphertext if the secret key is valid.

$Operation_{params,evk}(c_1, c_2) \rightarrow c$ *or* $\perp$: A binary operation is performed with the input ciphertexts and outputs a resultant ciphertext. If no more multiplication can be done with either c_1 or c_2, it outputs $\perp$. The operations supported depend on the scheme. Normally, they contain addition, multiplication, etc.

Figure 1. Algorithms in a Somewhat homomorphic encryption (SWHE).

Hybrid HE(Homomorphic Encryption) [18,19] provides a method of encoding a large amount of data on the existing SWHE and a method of quickly searching for given data among the encoded data; it easily finds the associated data with the searched data. Hybrid HE methods have been proposed in order to quickly find specific genome sequences in large amounts of genome sequence information. In [18,19], the genome sequencing data are stored in files of variations call format (VCF). Figure 2 shows the method used to create an encrypted database from the genome sequence data.

Figure 2. Process of creating an encrypted database in hybrid homomorphic encryption (HE).

The base sequence information includes all genotype information, such as chromosome number and nucleotide sequence position, and it is stored in a VCF file. A set of base sequence information is retrieved from the files of the VCF to create a database. Subsequently, the retrieved base sequence information is encoded into an element of a polynomial ring, which is treated as a unit storage of the database.

Table 1 shows the format of a VCF file that stores the base sequence information. In Table 1, in the VCF file descriptions, each line consists of (ch_i, pos_i, $SNPs_i$). ch_i is an identifier of a chromosome and it is a value in the range 1–22 X and Y. pos_i indicates the position information and it is a positive integer value. $SNPs_i$ contains the information of the REF base and ALT base sequence, respectively. The REF/ALT base sequences that are stored in $SNPs_i$ consist of a number of SNPs; SNP refers to one base and each base is one of A, T, G, and C. It is represented using two bits according to the following rules:

$$\mathrm{A} \mapsto 00,\ \mathrm{T} \mapsto 01,\ \mathrm{G} \mapsto 10,\ \mathrm{C} \mapsto 11$$

m_{SNP} refers to the maximum number of REF/ALT base sequences that can constitute a key in a unit data stored in the database and, thus, be compared with query data. If m_{SNP} is two, then the key part is made with two bases from REF, ALT, or both. Therefore, as m_{SNP} increases, the number of alleles that is used as a key in the REF and ALT bases is increased.

Table 1. VCF file format of hybrid HE [18].

CHROM.	POS	REF	ALT
1	161235340	G	A
1	161235596	C	T
1	161235657	G	T
1	161235981	G	A
1	161237503	-	TTTTTTGT
1	161239028	AG	-
1	161239142	A	G
1	161239346	G	T
1	161239470	C	T
1	161239788	-	AA
1	16123978	C	T
1	161240641	TGAT	-

In Table 1, we explain how to encode each line in a VCF file. Each line in the VCF file is converted to a (d_i, α_i) pair. Each d_i is calculated while using ch_i and pos_i values, and α_i is calculated using the $SNPs_i$ value, for $i = 1, \ldots, N$, where N is the size of the database, as follows:

$$(ch_i,\ pos_i) \mapsto d_i = ch_i + 24 \times pos_i \tag{1}$$

$$\alpha_i = 2^{l_{alt}} \times \alpha_i^{ref} + \alpha_i^{alt} \tag{2}$$

In Equation (2), l_{alt} indicates the maximum bit length of the ALT sequences in a row $SNPs_i$ sequence. If we define l_{ref} as the maximum bit length of the REF sequences in a row, we can say that $l_{SNP} = l_{alt} + l_{ref.} \times \alpha_i^{ref}$ and α_i^{alt} are the integer encodings of the REF and ALT base sequences in row #i, respectively. The converted (d_i, α_i) value of Equations (1) and (2) is embedded as a term of an element in a polynomial ring. Thus, if we suppose that $l_{SNP}/2 \leq m_{SNP}$, we can represent a database as an element in a polynomial ring if the number of values inserted is less than the maximum possible degree of the element, as follows:

$$DB(x) = \sum_i \alpha_i X^{d_i} \tag{3}$$

Let us explain how a query and search operation is created. To search a (d, α) pair in $DB(x)$, we first generate a query polynomial $\mathrm{Q}(x) = \mathrm{X}^{-d}$. Subsequently, the polynomial is multiplied with $DB(x)$ over the defined polynomial ring. The constant term of the

multiplication result is then extracted. Let this be β. Subsequently, we compare β with α. If both of them are the same, (d, α) is in $DB(x)$, or else it is not considered.

Owing to [18], we can perform such a search operation with an encrypted database and an encrypted query. After encrypting $DB(x)$ to Enc($DB(x)$) and $Q(x)$ to Enc($Q(x)$), it is possible to perform multiplication with both ciphertexts in order to obtain the ciphertext of $DB(x) \times Q(x)$. Thus, we can obtain $DB(x) \times Q(x)$, which can be easily transformed to the search result after decrypting the ciphertext.

Unfortunately, hybrid HE cannot be applied directly as a means for our purpose; the reasons are, as follows. The first is the problem of data size. Hybrid HE deals with data of size less than 30 bits [18]; it does not describe how to deal with data of longer bit length, such as RRN. Second, if the number of data increases, to search for a large amount of data, we need to use more than a single polynomial $DB(x)$ because of the limitation of the maximum degree of polynomials in the ring. The database of multiple encrypted polynomials must be efficient without wasting space, and they should be wisely organized for efficient search operations. However, in [18], an extension of database to deal with data of longer length and more numbers is not considered.

2.2. Related Work

In this subsection, we examine previous research on encrypted data retrieval using HE that is based on their comparison operations. In general, in HE, comparing whether the encrypted data are identical to the queried data is known to be inefficient, because it requires a high circuit depth [21–23].

Togan and Pleşca [24] designed a model for comparison between hidden plaintexts of binary strings in FHE(Fully Homomorphic Encryption)-based encrypted data, and implemented it using the HElib [25–27] library. In Reference [24], the following method was applied to compare two ciphertexts, including n-bit length values.

$$\overbrace{x_{n-1}\dots x_l}^{msb(X)}\overbrace{x_{l-1}\dots x_0}^{lsb(X)} > \overbrace{y_{n-1}\dots y_l}^{msb(Y)}\overbrace{y_{l-1}\dots y_0}^{lsb(Y)} \Leftrightarrow (msb(X) > msb(Y)) \vee ((msb(X) = msb(Y) \wedge (lsb(X) > lsb(Y)) \quad (4)$$

A comparison operation using Equation (4) is performed, as follows. To find the larger of the two input ciphertexts, X and Y, the values are compared in order from the most significant bit to the least significant bit. The result of the comparison operation for each bit is stored as a ciphertext and, as a last step, the comparison result of x_0 and y_0 bits and the stored result ciphertexts are operated together to calculate the final comparison result. When the corresponding model is implemented with FHE and a comparison operation is performed on an 8-bit value, an operation to find the equality of the data takes approximately 5 s and the comparison operation takes approximately 10 s in a conventional PC setting. In addition, it was confirmed that, as the bit length of the encrypted data increases, the time that is required for the operation also increases linearly.

Carlton [28] proposed a comparison protocol when integers were encrypted with FHE; he implemented his method using SageMath [29] in Python. In Reference [28], the threshold function was defined and applied, as shown in Equation (5).

$$f(m_1, m_2) = \begin{cases} 0, & | \ m_1 + m_2 \geq t \\ m_1 + m_2, & | \text{otherwise.} \end{cases} \quad (5)$$

In Equation (5), when an addition operation is performed on ciphertexts m_1 and m_2, the result becomes 0 if it exceeds the threshold t. Therefore, if the operation result of m_1 + m_2 is 0, then it can be seen that the actual value is t or more. The protocol proposed in [28] provides a comparison function for integer data, not binary numbers, and takes approximately 0.2 s to execute. However, the protocol is difficult to use in HE, which makes it difficult to apply the threshold.

Bonte, C. and Iliashenko, I. worked to increase the efficiency of these comparisons in their pattern matching study [30]. They approximate the terms of the OR gate that make

up the comparison operation of pattern matching while using a low degree multilinear polynomial that is based on the Razborov–Smolensky method [31,32]. This approach is efficient because the comparison circuit can have a multiplication depth that is independent of the length of the pattern to be matched. However, the application of this technique was not considered, because the RRN to be searched in this study is relatively short.

In Laine, K's work [33], a method was proposed and implemented to determine whether to match the encrypted string based on SEAL [34]. It employs cuckoo hashing [35] to compare strings of arbitrary length. This method takes approximately 225 s to compare 10,000 bits of data. However, because of the limitation of SEAL, there is a disadvantage that the number of operations for comparing whether or not data are identical is limited.

In addition, studies have been proposed to modify the comparison operation according to the purpose of the application or to efficiently perform the comparison operation on multiple data [36–38]. The study [36] proposed an application of FHE for expressing floating-point numbers, such as IEEE 754, and designed a specialized comparison operation while considering the sign, exponent, and fraction of real numbers in floating-point format. In the study [37], a method for efficiently finding MAX values among multiple data was proposed through parallel processing. For data, each comparison operation with other values is performed in parallel and logical operations are performed on the results of the comparisons to determine which value is MAX. In the study of [38], comparison operation is performed to determine the case/special character of ASCII code and check the string length, and an encoding method that is suitable for this was developed. It extends the ASCII encoding to the slot inside the ciphertext in a form that is easy to compare.

2.3. The Structure of the Korean Residence Registration Numbers (RRNs)

This subsection describes RRNs that will aid in an understanding of this paper. All Korean citizens must get an RRN immediately after birth and all citizens have a unique 13-digit RRN. The inventors of RRN in Korea used the social security number system used in the United States as a reference. Because every citizen has a unique RRN, the RRN is used by many Korean websites and smartphone applications as a weak secret information source for identification and authentication. Therefore, when an individual's RRN is exposed, the damage can be greater than that of other sensitive information, and a study of [3,4] to actually collect RRN has also been proposed.

The structure of the RRN to be described in this subsection follows the notation of Choi, D. et al. [3]. RRN consists of 13 decimal digits, such as "ABCDEF-GHIJKLM". The meaning of the numbers is as follows:

- AB: The last two digits of the holder's birth year.
- CDEF: The holder's birth month/date.
- G: Determined based on the year of birth and the sex of the holder as follows.
  - 9 (0): Male (female) born 1800~1899
  - 1 (2): Male (female) born in 1900~1999
  - 3 (4): Male (female) born in 2000~2099
- HI/JK: The code corresponding to (state, county)/(city, village) of the holder's birth region is provided.
- L: The order of registration at the registration office in the same region on the same date, counted by gender.
- M: Checksum value that is determined by the different digits of the RRN according to the formula

$$M = (11 - \{(2 \times A + 3 \times B + 4 \times C + 5 \times D + 6 \times E + 7 \times F + 8 \times G + 9 \times H + 2 \times I + 3 \times J + 4 \times K + 5 \times L) \bmod 11\}) \bmod 10.$$

3. Search Algorithm for Encrypted Data

In this work, we propose a method for efficiently storing and searching a large number of encrypted RRN data. For this, we modify the hybrid HE and design it to store the RRN data efficiently.

3.1. Overview of the Proposed Work

Figure 3 presents an overview of the proposed scheme. There are two actors in the system. One is the user who generates private, evaluation, and public keys using hybrid HE [18,19]. The public and evaluation keys are sent to the other actor, the server, which manages the encrypted database and performs query operations with the querying ciphertexts from users. The proposed scheme supports two operations: the database setup and query operation. Figure 3a shows the database setup operation. It consists of three steps. The first step is the formatting of the data. Each RRN is formatted to a number of (d, α) pairs, depending on the number of databases (nDB) used. For simplicity, we assume that an RRN is formatted to a single (d, α) pair. Subsequently, each (d, α) pair is embedded into a polynomial, which makes the coefficient of the dth term as α.

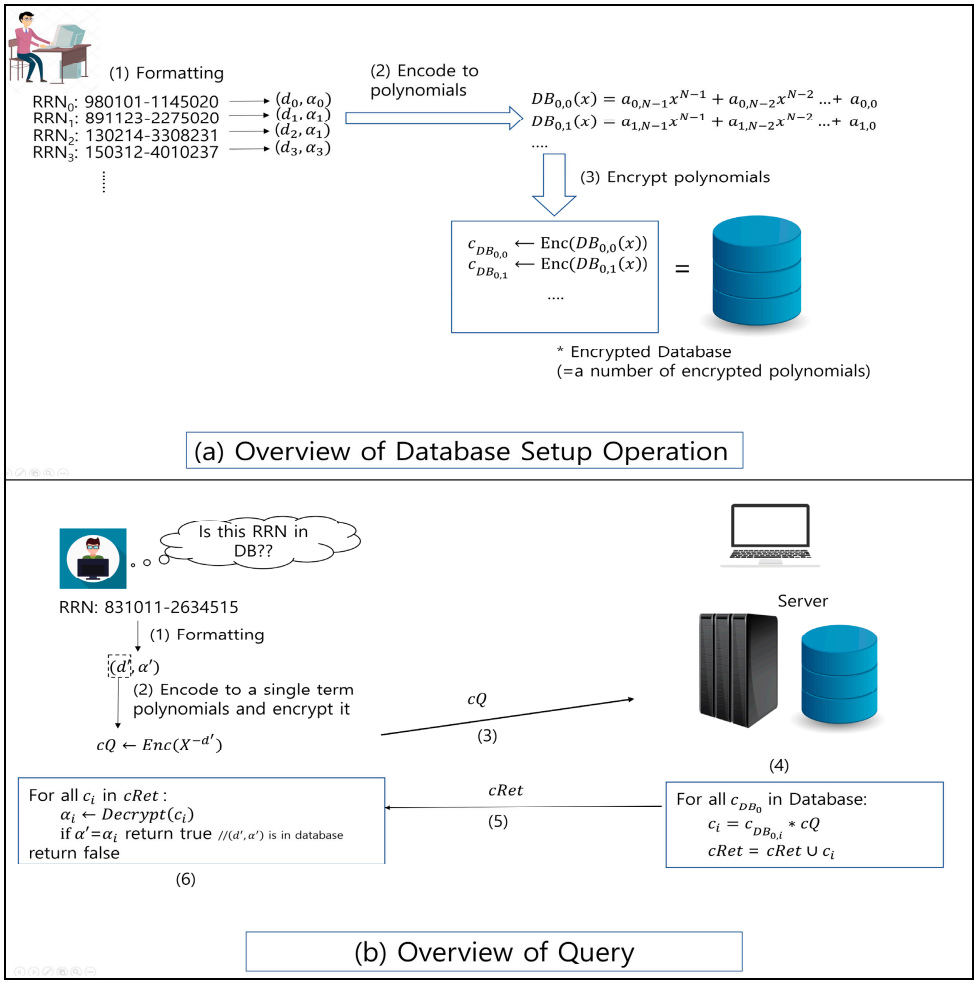

Figure 3. Overview of the proposed scheme.

It is possible that two different RRNs can be of the same ds, but different αs. In this case, a new polynomial is generated, and the later RRN is embedded into the newly generated polynomial. The final step is encryption, where the polynomial is encrypted using the encryption scheme that is specified in [18]. The encrypted database is constructed as a result of encryption. Figure 3b shows the query operation. In the database setup step, the RRN to be queried is formatted and then encoded to a single term polynomial. Afterwards,

the polynomial is encrypted and sent to the server. Subsequently, the server performs a multiplication operation between the delivered query polynomial and each of the stored polynomials in the database without decrypting them. Because the scheme presented in [18] is a homomorphic encryption, the result of the multiplication of the ciphertext is a ciphertext that contains the multiplication result of the two plaintext polynomials hidden in the two input ciphertexts. The result of multiplication is sent back to the user; then, the user performs decryption with the received ciphertexts. It checks that there is α' (one chunk from the input RRN being queried) among the decryption results. If so, the queried polynomial is in the database, or else, it is not.

Unfortunately, the overview does not explain the proposed scheme exactly. Because of the limitation in shown in [18], an RRN cannot be represented with a single (d, α) pair. A single pair can only contain 22 bits; thus, we propose a method to encode an RRN with multiple (d, α) pairs and make the database setup and the query operation efficient.

We use two approaches to perform this operation. The first approach is to use multiple pairs (d, α_0), $(d, \alpha_1), \ldots, (d, \alpha_{n-1})$, where d is fixed but α_is is different. Because we can only contain one α_i of the dth term in a polynomial, we need at least n polynomials to store one RRN. In addition, if two RRNs need to be stored, but they have the same d, the number of polynomials needed increases by n. Another approach is to use multiple ds. In this case, the number of polynomials needed can be less than the first case, because both the d-part and α-part can be used to store the RRN. However, the number of ciphertexts needed for a query is increased because the number of ds used is more than one. In the next subsection, we provide the details of each step of the two approaches. We assume that the number of ds used is denoted as *nDB*, the number of databases, and the number of αs used is the number of columns, *nCol*.

From a privacy point of view, in the query process (4) of Figure 3b, the result of the data items to be searched is stored as a constant term of polynomial α_is and the user has to check the constant terms. Therefore, there are risks of leakage of server database information, as follows: First, some of the internal information of the server database might be included in coefficients due to the computation result of (4). Second, the user can obtain the data α_i of another user who shares the same value as his/her d' as the query result.

The solutions to these problems are covered in the study [18], and the same approach can be applied to our proposed method. First, we apply the technique [39] of converting the polynomial RLWE encryption to the constant LWE encryption, which is used in the bootstrapping process of FHE, in order to extract the constant term of the search result. The server can extract constant terms in encrypted form by applying this conversion procedure to c_is, and prevent the leakage of unqueried information of the coefficients. Next, the one-way hash function can be applied for safe comparison in (6) of Figure 3b. If the hashed values of the data items are stored in the sever database, the user performs a comparison of the hashed result in (6), so that the user can only check the equality without knowing the information stored in the database. Additionally, exposure to indirect information through checking the number of c_is in the query result *cRet* can be prevented in a way that the server includes arbitrary values that are encrypted in *cRet*.

3.2. Data Formatting for RRN

We provide details of how to format an RRN into a set of (d, α) pairs in this subsection. We can choose different formatting strategies depending on the *nDB* value, as shown in Figure 4. Figure 4a shows the case where *nDB* is one. In this case, three αs are required to represent one RRN. It is to be noted that both d and α can contain up to 11 bits. However, to represent whether there is a value in the bit or not, the most significant bit of α is used as a flag bit. Thus, each of three-digit numbers in an RRN are covered by each α. In addition, the 3–6th digits of an RRN represent the birthday of the holder. Thus, this number is less than 2047. Therefore, we can contain those digits in d. Figure 4b shows the case of $nDB = 2$. In this case, there are two ds; hence, they contain up to 22 bits. Because the first digit in the second part of RRN can be only one of 1–4, all seven digits can be contained in two ds.

Figure 4c represents the case where $nDB = 4$. In this case, all the digits in an RRN can be contained in the four ds, and we do not have to put any digit in the α part. However, the most significant bit of α is set to one to represent whether a given d value is used. Figure 4d depicts the relation between nDB and the number of α values needed to represent an RRN. From the table, we can see that the maximum nDB that we need to consider is four.

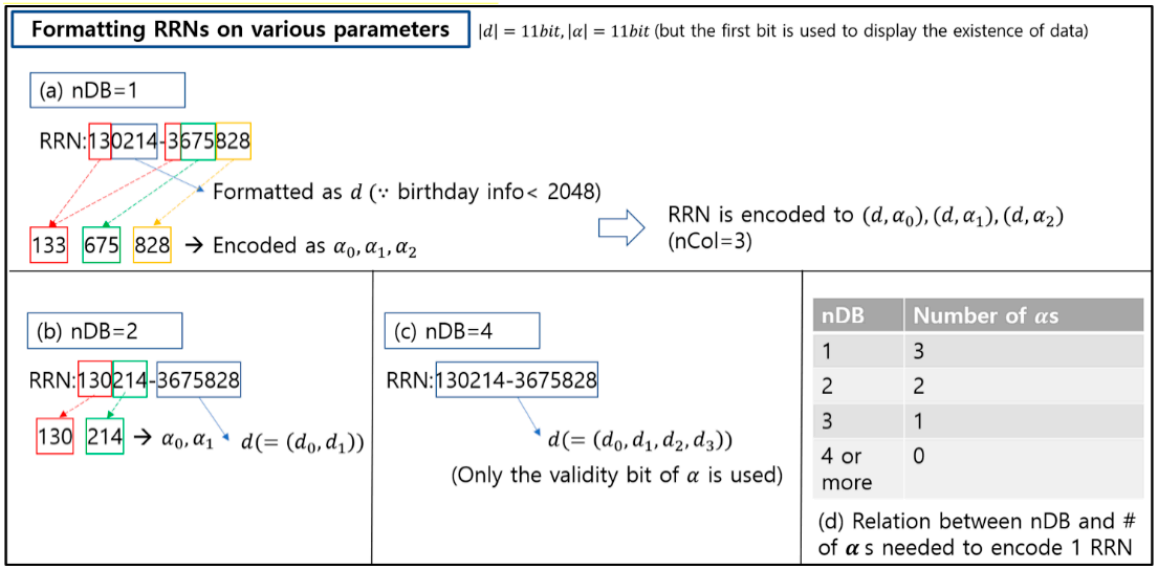

Figure 4. Formatting residence registration numbers (RRNs) on various parameters (when $nDB = 1, 2, 3, 4$).

3.3. Data Encoding and Building Databases

After formatting, the data are encoded to organize databases or to make a query. We focus on making databases with a large number of formatted RRNs, as the case of making a query is encoding a single formatted RRN. The encoding comprises two steps. The first step is embedding the formatted RRNs into polynomials. Figure 5 depicts this step. It shows that each (d, α) pair constitutes a single term in a polynomial. In addition, at $d_0 = d_1$, when the two pairs of formatted (d_0, α_0) and (d_1, α_1) are encoded, they are stored in two separate polynomials. Figure 5a explains the case when a single d is used with multiple αs. In this case, depending on the number of αs used, the number of polynomials that is newly created when two RRNs collide with regard to their d values is considered. Figure 5b deals with the case where two ds are used to format an RRN value. As fewer numbers of αs are mapped to a single d, the number of polynomials created in a collision, which is, the event that any of d in two RRNs is matched occurs is decreased.

After the polynomials are generated, they are encrypted using the scheme presented in [18]. The encrypted polynomials are stored in the server as databases. It is to be noted that, even if they are encrypted, we can perform search operations with them if we have a ciphertext to be queried.

3.4. Search Protocol

We provide the details of the search protocol shown in Figure 6. The query polynomial is encrypted with the public key, which is used to encrypt the databases in the server, as shown in step (2) of Figure 6. The server performs the operations in step (4), and the results are returned to the database. Owing to [18], the multiplication result can be much shorter than a ciphertext in the database. Thus, the size of the query result can be processed online. The user performs step (6) after the query result *cRet* is delivered. If the decryption result is matched to the αs generated from the queried RRN, it concludes that the queried RRN exists in the database.

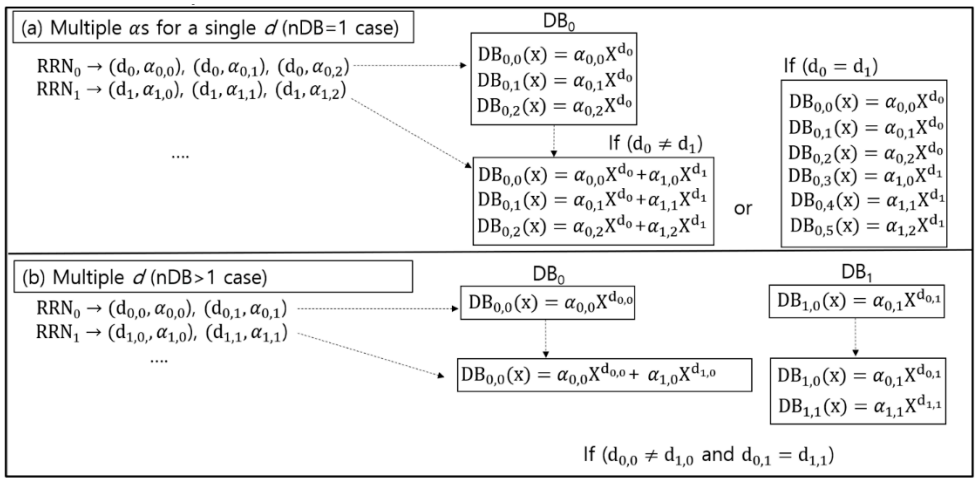

Figure 5. Encoding formatted data to polynomials (when $nDB = 1, 2$).

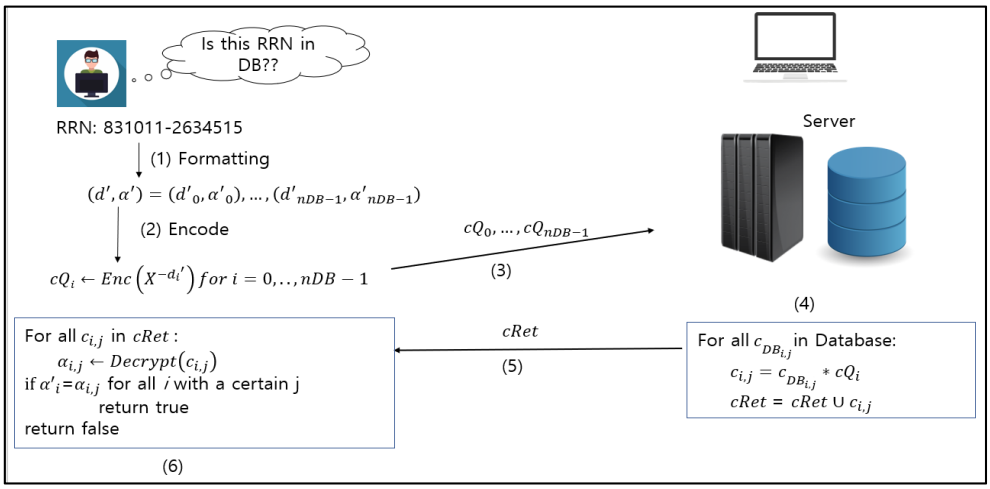

Figure 6. Search protocol.

4. Performance Evaluation

In this section, we evaluate the performance of the proposed scheme with real data and then compare it with that of other methods. For this, we implement the search operation on all of the methods and perform the experiment in the same environment as the proposed method. The compared methods are implemented with open source libraries, such as HElib [24–26], HEAAN [40,41], nuFHE [42], and Python-Paillier [43]. We measure the execution times of operations, the size of the database to store the same size of data, the size of the search query, and the resultant ciphertexts of the search operation. For the measurement, the proposed method encodes a pair of d as a search keyword and a data item α to be searched while using each encryption algorithm. However, it is impossible for other encryption algorithm to construct a database using a polynomial utilizing α as coefficient and d as degree, so, for other encryption algorithms, d and α are stored in bit encoding, except for Paillier ciphers, which are in integer encodings. Thus, parallel processing of data sharing keywords and configuration of multiple DBs were not considered, which are based on the polynomial encoding.

4.1. Search Algorithms for Other Methods Compared

For performance comparison, we implemented the FHE-based encryption data search algorithm that is described in Figure 7 using various homomorphic encryption libraries of HElib, HEAAN, and nuFHE. The performance of the other methods was then compared with the proposed method. The search is performed by subtracting the data to be searched for in

each of the data stored in the database, and multiplying all of the result values. As a result, if there are data to be searched in the database, the result ciphertext encrypting 0 is returned and, if there is no data, the result ciphertext encrypting a non-zero value is returned.

```
cQ: It is a ciphertext that encrypts the query Q for search, cQ←Enc(Q).
function fhe_search(cQ){
  lineNum←0;
  while(infile!=eof)() do
      plaindata←infile.read(line);    //Read from the database file
      cDB[lineNum++]←Enc(plaindata); // Data encryption
  end while

  for(i←0; i<lineNum; i++) do
     cSub[i]←sub(cDB[i], cQ); // Subtraction operation
  end for
  for(i←0; i<lineNum-1; i++) do
     cSub[i+1]←mult(cSub[i], cSub[i+1]); // Multiplication operation
  end for
  cRes = cSub[lineNum-1];
Return cRes;
```

Figure 7. Pseudo code of search operation in FHE.

We also implemented a search algorithm that could work with the Paillier cryptosystem, which is an additive homomorphic encryption algorithm, as shown in Figure 8. To implement this, we used Python-Paillier, an implementation of the Paillier cryptosystem [9] with Python language. Unlike the FHE schemes, the Paillier scheme does not support the multiplication operation of two ciphertexts. Here, the ciphertext obtained by subtracting the data to be searched and the data stored in the database are returned as a result. If there are data to be searched in the database, there is a 0 in the result ciphertext; if there are no data, there is non-zero value in the result ciphertext.

```
cQ: It is a ciphertext that encrypts the query Q for search, cQ←Enc(Q).
function paillier_search(cQ){
lineNum←0;
  while(infile!=eof)() do
      plaindata←infile.read(line);    //Read from the database file
      cDB[lineNum++]←Enc(plaindata); // Data encryption
  end while

  for(i←0; i<lineNum; i++) do
     cRes[i]←sub(cDB[i], cQ); // Subtraction operation
  end for
Return cRes;
```

Figure 8. Pseudo code of search operation in Paillier Cryptosystem.

4.2. Evaluation Environment

For the performance evaluation of the search algorithm, Table 2; Table 3 show the hardware environment and parameter settings in which the algorithm was performed.

Table 2. Performance Evaluation Environment.

	Hardware Specification
CPU	Intel i7-6700
RAM	16 GB
OS	Linux Ubuntu 18.04
GPU	GeForce GTX 1070

Table 3. Parameters used for each library.

Library.	**Parameters**
Hybrid HE (for proposed algorithm)	security parameter = 128 bit (according to [44])
HElib [24–26]	R = 1, p = 2, r = 1, d = 1, c = 2, L = 0, s = 0, m = 65537, security parameter = 80, slots = 2048
HEAAN [32,33]	*log*N = 15, *log*P = 23, *log*q = 29, *log*Q = 620, *log*T = 2, slots = 2048
nuFHE [34]	lamda, λ = 80, slots = 2048
Python-Paillier [35]	BASE = 16

4.3. Performance Evaluation of the Proposed Algorithm

In this subsection, the execution time of the proposed method implemented on the environment that is shown in Table 2 and the size of the ciphertext are compared for various *nDB* environments. First, the execution times of each operation are compared, and then the size of the ciphertext is compared.

4.3.1. Comparison of the Execution Time

Encryption Execution Time: when the number of databases is 100,000, one million, or 10 million, the time that is required to encrypt the database where *nDB* is from 1 to 4 is shown in Figure 9a. It can be seen that, when the number of databases is 100,000, 1 million, or 10 million, encryption is performed fastest when the *nDB* is three.

Query Encryption Execution Time: when the number of databases is 100,000, one million, or 10 million, the time that is required to encrypt queries where *nDB* is from 1 to 4 is shown in Figure 9b. Query data are generated as many as *nDB*, and it can be seen that the execution time for query encryption increases as the *nDB* increases.

Search Execution Time: when the number of data in the database is 100,000, one million, or 10 million, the time that is required to search for data in the database where *nDB* is from 1 to 4 is shown in Figure 9c. Similar to the encryption execution time, when the number of data in the database is 100,000, one million, or 10 million, it can be seen that the data search is performed fastest when the *nDB* is three.

Result Decryption Time: when the number of databases is 100,000, one million, or 10 million, the time required to decrypt the result ciphertext where *nDB* is from 1 to 4 is shown in Figure 9d. It can be seen that when the number of data in the database is 100,000, one million, or 10 million, and when the *nDB* is three, the decryption is performed faster with a slight difference than in other cases.

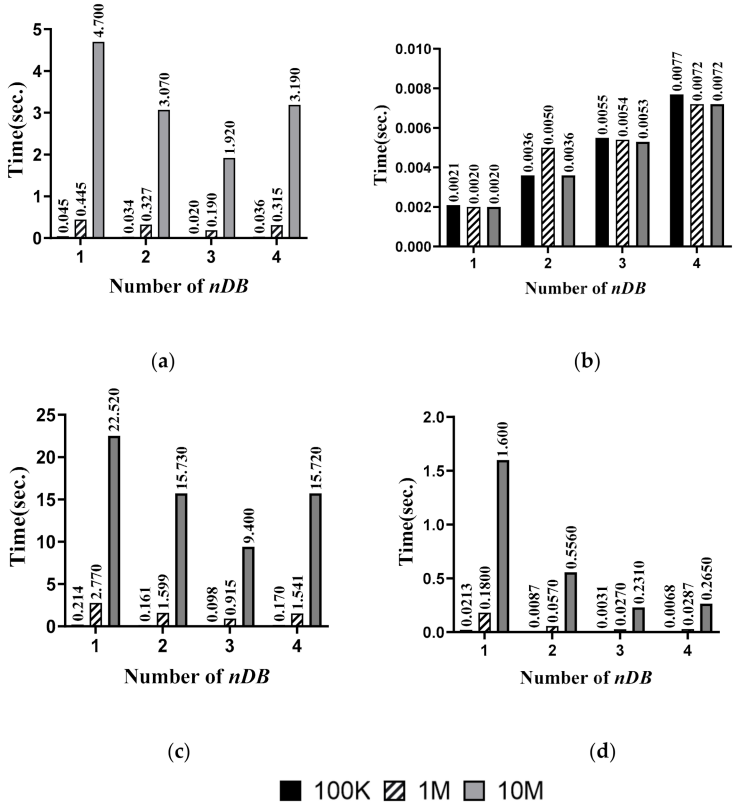

Figure 9. Comparison of the execution time: (**a**) performance of the database encryption process; (**b**) performance of the query encryption process; (**c**) performance of search operations; and, (**d**) performance of decrypting the result ciphertext.

4.3.2. Comparison of the Size of the Result Ciphertexts

Size of Ciphertext for Database Representation: when the number of databases is 100,000, one million, or 10 million, the size of the ciphertext for the database where *nDB* is from one to four is as shown in Figure 10a. When the number of data in the database is 100,000, one million, or 10 million, and, when the *nDB* is three, the size of the database is the smallest.

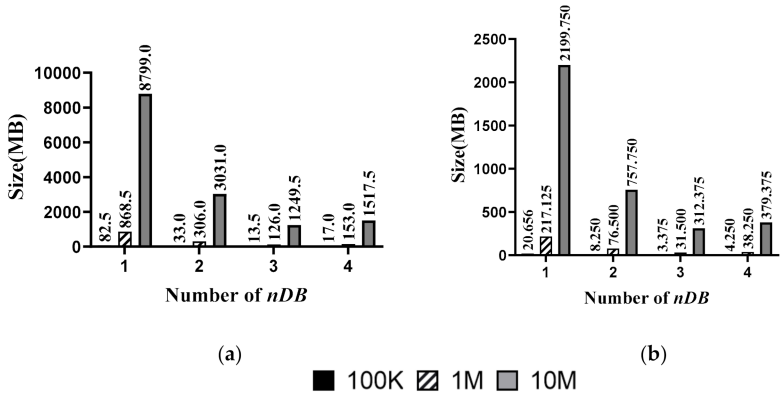

Figure 10. Comparison of the size of the result ciphertexts: (**a**) size of database ciphertexts; (**b**) size of result ciphertext.

Ciphertext Size for Query Result: when the number of databases is 100,000, 1,000,000, or 10 million, the size of the result ciphertext where *nDB* is from 1 to 4 is as shown in Figure 10b. Similar to the size of the database, when the number of data in the database is 100,000, one million, or 10 million, it can be seen that the size of the result ciphertext is the smallest when the *nDB* is three.

4.4. Performance Comparison with the Search Algorithm to Be Compared

In this subsection, we compare the performance of the proposed algorithm with an optimal *nDB* setting, which we found as a result in Section 4.3, to the other methods that were previously mentioned. Comparisons are only performed when the number of data in the database is 10,000, owing to performance limitations of the other methods. The execution time and size of the ciphertext are compared to execute the search operation.

Comparison of Search Algorithm Execution Time

Search Execution Time: when the number of data points is 10,000, the time that is required for the search is shown in Figure 11. Among the methods to be compared, Python-Paillier is the fastest; however, the performance is approximately 43.9 times worse than the proposed algorithm.

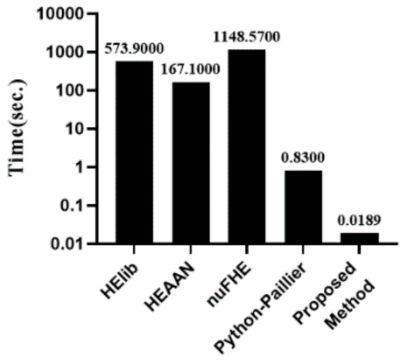

Figure 11. Performance comparison of search operations.

Search Result Decryption Time: Figure 12 shows the time that is required for decoding. In the search algorithm for performance comparison, it can be seen that nuFHE performs the operation the fastest; however, it also takes 77.5 times longer than the proposed algorithm.

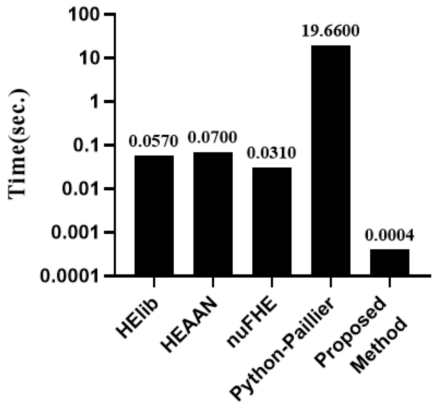

Figure 12. Performance comparison on decrypting the result ciphertext for a search query (s).

Size of Ciphertext for Database Representation: when the number of databases is 10,000, the size of the encrypted database of each method is shown in Figure 13. It can be seen that the ciphertexts of nuFHE are generated in the smallest size. The proposed algorithm is 2.5 MB, 250 bytes per record, and it can be considered to be practical.

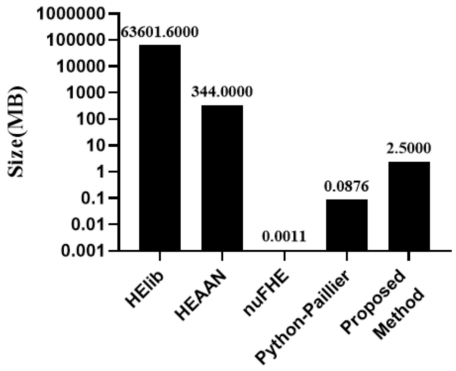

Figure 13. Size comparison of encrypted database for 10,000 records.

Ciphertext Size for Query Result: Figure 14 show the size of the ciphertext in the search query result. It can be seen that nuFHE is generated in the smallest size. The result ciphertext size of the proposed algorithm is 625 kB, which is affordable when considering the available memory/storage size of the current computer system.

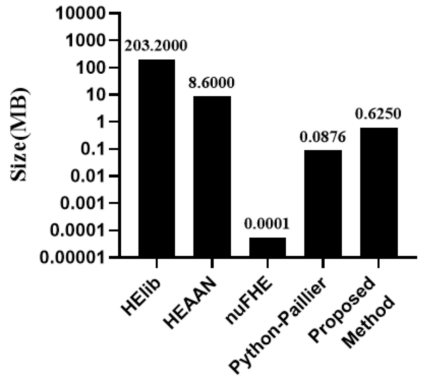

Figure 14. Size comparison on the resultant ciphertext for a search query.

4.5. Review of the Evaluation Results

In this subsection, based on the results of Sections 4.3 and 4.4, once again we discuss the *nDB* value setting of the proposed method and its search performance. Because the performance of the search operation differs according to the *nDB* setting, the optimal *nDB* value for our environment was found in this study. It was found that in a general PC environment, data can be searched within 12 s, even in a database in which 10 million RRN data are contained. Table 4 summarizes the timings and parameter values for the fastest search for 100,000, one million, and 10 million RRN data.

Table 4. Performance of the proposed scheme on the best parameter.

# of Data	Run Time (s)				*nDB*
	DB Encryption	Query Encryption	Search	Decryption	
100,000	0.02	0.005	0.098	0.0031	3
1 million	0.19	0.005	0.91	0.027	3
10 million	1.92	0.005	9.40	0.231	3

When the data are 100,000, one million, or 10 million, the speed of the search operation is best when the *nDB* is three; the search operation for 10 million data points can be performed within 12 s. From the result, we can see that the search operation can be

performed up to 49,000 times faster than the search operation that was provided by the previous HE methods [24–26,40–43].

Table 5 summarizes the size and parameters of the ciphertext encrypted with the smallest size as a result of comparing the result ciphertext size of the search operation with the database, in which 100,000, 1,000,000, and 10 million RRN data are encrypted. This result can be used to reduce the overall search performance and ciphertext size values.

Table 5. Ciphertext size of the proposed scheme on the best parameter value.

# of Data.	Size of Ciphertext (MB)		*nDB*
	DB	Query Result Ciphertext	
100,000	13.5	3.375	3
1 million	126	31.5	3
10 million	1249.5	312.375	3

In the search for RRNs, it can be seen that, when there are 100,000, one million, and 10 million data, the database and the resulting ciphertexts are created with the smallest size when the *nDB* value is three.

5. Discussion

In this section, we discuss some of the issues that need to be resolved or were not clearly addressed in the previous sections. The first issue is the possibility of a false positive when multiple ds are used. In order to resolve this issue, we employ the unused αs to store the hash of the formatted (d, α) pairs for an input RRN. After decrypting the query result, once a candidate matching is found, we extract the hash value from the unused αs in the decrypted one and then check whether the stored hash can be computed from the queried input. Owing to the one-wayness property, the probability of the false positive will be low if we use the hash algorithm whose output bit is around 20 bits.

Another issue is the possibility that this approach can be extended to store an arbitrary type of data. It is possible to store any binary data. The database in this paper acts as a so-called dictionary structure, where the data are organized as (key, value) pairs: ds act as keys and αs are the corresponding values. Therefore, any binary string can be stored in the proposed database structure.

6. Conclusions

In this work, we proposed a new search method over encrypted data, which complements the existing hybrid HE and helps to deal with a large number of RRNs. In the existing hybrid HE, there is a limit where a single database encoded and encrypted in a polynomial ring format can only store data of up to 11 bits as a key and value, respectively. Therefore, we designed and implemented an extended method to more efficiently store RRN data by expanding the bits of data that can be stored using multiple databases from the basic hybrid HE. Through experiments, we confirmed that the search operation can be performed up to 49,000 times faster than the search operation that was provided by the previous methods [24–26,40–43].

The result of this work solves the problem of the limitation on the size of the encrypted database for search operations because of the slow operation speed, which is one of the well-known problems in FHE-based solutions for searching the encrypted data. Based on this, it is expected that the proposed method can be practically used in the industrial field dealing with personal information data, such as credit information evaluation and personal health, which requires privacy protection.

Author Contributions: Conceptualization, E.J. and Y.L.; methodology, E.J.; software, E.J. and P.K.; validation, P.K.; formal analysis, Y.L.; data curation, E.J.; writing—original draft preparation, E.J. and

P.K.; writing—review and editing, Y.L.; visualization, P.K.; supervision, Y.L.; project administration, Y.L.; funding acquisition, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2019R1A2C4069769).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Malandrino, D.; Petta, A.; Scarano, V.; Serra, L.; Spinelli, R.; Krishnamurthy, B. Privacy awareness about information leakage: Who knows what about me? In *Proceedings of the 12th ACM workshop on Workshop on Privacy in the Electronic Society*; Association for Computing Machinery: New York, NY, USA, 2013; pp. 279–284.
2. Kim, S.; Lee, K. A Comparative Study on Reforming the Resident Registration Number. *J. Korea Inst. Inf. Secur. Cryptol.* **2015**, *25*, 673–689.
3. Choi, D.; Lee, Y.; Park, Y.; Kim, S. Estimating Korean residence registration numbers from public information on SNS. *IEICE Trans. Commun.* **2015**, *98*, 565–574. [CrossRef]
4. Kim, H.; Park, K.; Choi, D.; Lee, Y. Estimating resident registration numbers of individuals in Korea: Revisited. *KSII Trans. Internet Inf. Syst.* **2018**, *12*, 87–108.
5. Kim, J. A Study on Improvement method of designation criteria for Personal Proofing Service Based on Resident Registration Number. *J. Korea Soc. Digit. Ind. Inf. Manag.* **2020**, *16*, 13–23.
6. Gentry, C. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 2009. Available online: http://crypto.stanford.edu/craig (accessed on 3 December 2020).
7. Brakerski, Z.; Gentry, C.; Vaikuntanathan, C. (Leveled) fully homomorphic encryption without bootstrapping. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, Academic Medicine, Cambridge, MA, USA, 8–10 January 2012; pp. 309–325.
8. Brakerski, Z. Fully homomorphic encryption without modulus swithing from classical gapsvp. In *Advances in Cryptology–Cryptogr*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7417, pp. 868–886.
9. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Lecture Notes in Computer Science*; Advances in Cryptology—EUROCRYPT '99; Stern, J., Ed.; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1592.
10. Rivest, R.; Adleman, L.; Dertouzos, M. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*; Academic Press Inc.: Cambridge, MA, USA, 1978; pp. 169–180.
11. Brickell, E.F.; Yacobi, Y. On privacy homomorphisms. In *Lecture Notes in Computer Science*; Advances in Cryptology-EUROCRYPT '87; David, C., Wyn, L.P., Eds.; Springer: Berlin, Germany, 1987; Volume 304, pp. 117–126.
12. Frieze, A.M.; Hastad, J.; Kannan, R.; Lagarias, J.C.; Shamir, A. Reconstructing truncated integer variables satisfying linear congruences. *Siam J. Comput.* **1988**, *17*, 262–280. [CrossRef]
13. Brickell, E.F.; Odlyzko, A.M. Cryptanalysis: A survey of recent results. *Proc. IEEE* **1988**, *76*, 578–593. [CrossRef]
14. I Ferrer, J.D. A new privacy homomorphism and applications. *Inf. Process. Lett.* **1996**, *60*, 277–282. [CrossRef]
15. Goldwasser, S.; Micali, S. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Proceedings of the 14th symposium on Theory of Computing, San Francisco, CA, USA, 5–7 May 1982; pp. 365–377.
16. ElGamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472. [CrossRef]
17. Benaloh, J. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*; Springer: Berlin/Heidelberg, Germany, 1994; pp. 120–128.
18. Kim, M.; Song, Y.; Cheon, J.H. Secure searching of biomarkers through hybrid homomorphic encryption scheme. *BMC Med. Genom.* **2017**, *10*, 42. [CrossRef] [PubMed]
19. Kim, M.; Song, Y. Implementation of Secure Searching of Biomarkers. Available online: http://github.com/amedonis/HybridHE (accessed on 3 December 2020).
20. Lyubashevsky, V.; Peikert, C.; Regev, O. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–23.
21. Cheon, J.H.; Kim, M.; Kim, M. Search-and-compute on encrypted data. In *Lecture Notes in Computer Science International Conference on Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 142–159.
22. Kim, M.; Lauter, K. Private genome analysis through homomorphic encryption. *Bmc Med Inform. Decis. Mak.* **2015**, *15*, 1–12. [CrossRef] [PubMed]
23. Cheon, J.H.; Kim, M.; Kim, M. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 188–199. [CrossRef]
24. Togan, M.; Plesca, C. Comparison based computations over fully homomorphic encrypted data. In Proceedings of the IEEE 2014 10th International Conference on Communications (COMM), Bucharest, Romania, 29–31 May 2014; pp. 1–6.
25. Halevi, S.; Shoup, V. Algorithms in HElib. In *Proceedings of Advances in Cryptology—CRYPTO'14*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; p. 8616.

26. Halevi, S.; Shoup, V. Bootstrapping for HElib. In *Proceedings of Advances in Cryptology—EUROCRYPT 2015*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9056.
27. Halevi, S.; Shoup, V. HElib—An Implementation of Homomorphic Encryption. Available online: https://github.com/shaih/HElib/ (accessed on 3 December 2020).
28. Carlton, R.A. Secure integer comparisons using the homomorphic properties of prime power subgroups. MSc Thesis, The University of Western Ontario, London, ON, Canada, 2017.
29. Sage, T.S. Math, the Sage Mathematics Software System, Developers (Version 7.1). 2016, Volume 2020. Available online: http://www.sagemath.org (accessed on 3 December 2020).
30. Bonte, C.; Iliashenko, I. Homomorphic string search with constant multiplicative depth. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 105–117.
31. Razborov, A. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Math. Notes Acad. Sci. USSR* **1987**, *41*, 333–338. [CrossRef]
32. Smolensky, R. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*; Association for Computing Machinery: New York, NY, USA, 1987; pp. 77–82.
33. Laine, K. *String Matching on Homomorphically Encrypted Data*; Indian Institute of Science: Bangalore, India, 2017.
34. Chen, H.; Laine, K.; Player, R. Simple encrypted arithmetic library-SEAL v2. 1. In *International Conference on Financial Cryptography and Data Security*; Springer: Cham, Switzerland, 2017; pp. 3–18.
35. Pagh, R.; Rodler, F.F. Cuckoo hashing. *J. Algorithms* **2004**, *51*, 122–144. [CrossRef]
36. Moon, S.; Lee, Y. An efficient encrypted floating-point representation using HEAAN and TFHE. *Secur. Commun. Netw.* **2020**. [CrossRef]
37. Park, H.; Kim, P.; Kim, H.; Park, K.; Lee, Y. Efficient machine learning over encrypted data with non-interactive communication. *Comput. Stand. Interfaces* **2018**, *58*, 87–108. [CrossRef]
38. Kim, P.; Lee, Y.; Hong, Y.S.; Kwon, T. A Password Meter without Password Exposure. *Sensors* **2021**, *21*, 345. [CrossRef] [PubMed]
39. Ducas, L.; Micciancio, D. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 617–640.
40. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *Lecture Notes in Computer Science International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Cham, Switzerland, 2017; pp. 409–437.
41. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Bootstrapping for approximate homomorphic encryption. In *Advanced in Cryptology—EUROCRYPT*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 360–384.
42. nuFHE. Available online: https://github.com/nucypher/nufhe/ (accessed on 3 December 2020).
43. Python-Paillier. Available online: https://github.com/n1analytics/python-paillier/ (accessed on 3 December 2020).
44. Gentry, C.; Halevi, S.; Smart, N. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology-CRYPTO*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 850–867.