




Research Article

Secure Testing for Genetic Diseases on Encrypted Genomes with Homomorphic Encryption Scheme

Tan Ping Zhou ^{1,2,3}, Ning Bo Li,^{1,2} Xiao Yuan Yang ^{1,2}, Li Qun Lv,¹
Yi Tao Ding,¹ and Xu An Wang ¹

¹Key Laboratory of Network & Information Security under the People's Armed Police, Electronic Department, Engineering University of People's Armed Police, Xi'an 710086, China

²State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China

³State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Correspondence should be addressed to Xiao Yuan Yang; xyyangxyang@163.com and Xu An Wang; wangxazjd@163.com

Received 16 October 2017; Revised 3 January 2018; Accepted 22 January 2018; Published 20 March 2018

Academic Editor: Daniele Sgandurra

Copyright © 2018 Tan Ping Zhou et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The decline in genome sequencing costs has widened the population that can afford its cost and has also raised concerns about genetic privacy. Kim et al. present a practical solution to the scenario of secure searching of gene data on a semitrusted business cloud. However, there are three errors in their scheme. We have made three improvements to solve these three errors. (1) They truncate the variation encodings of gene to 21 bits, which causes LPCE error and more than 5% of the entries in the database cannot be queried integrally. We decompose these large encodings by 44 bits and deal with the components, respectively, to avoid LPCE error. (2) We abandon the hash function used in Kim's scheme, which may cause HCE error with a probability of 2^{-22} and decompose the position encoding of gene into three parts with the basis 2^{11} to avoid HCE error. (3) We analyze the relationship between the parameters and the CCE error and specify the condition that parameters need to satisfy to avoid the CCE error. Experiments show that our scheme can search all entries, and the probability of searching error is reduced to less than $2^{-37.4}$.

1. Introduction

Genes are the intrinsic nature of human health. All human life activities and physiological phenomena are directly related to the gene. Genome data can be used for a wide range of applications including healthcare, biomedical research, and forensics [1]. Gene sequencing technology is the core of the human genome project; the genome sequencing technology helps humans to better understand the whole life activities of cells and organisms, and it is also of great significance for the prevention and treatment of some diseases, such as cancer and genetic diseases.

Advances in high throughput technologies have made it increasingly affordable to sequence the human genome in various settings, ranging from biomedical research to healthcare [2]. Relevant data show that in 2000 the cost of whole genome sequencing for human is nearly \$3 billion, and by 2015, the cost of single genome sequencing is reduced to

less than \$1,000, and the sequencing costs for certain sites on the genome are lower.

The decline in genome sequencing costs has widened the population that can afford the cost of gene sequencing and has also raised concerns about genetic privacy. Genetic data can be widely used in healthcare, biomedical research and identification, and other fields, with a strong personal privacy characteristics. More and more businesses and individuals put the computing processing of genetic data to cloud services, but the current commercial cloud server does not fully guarantee the privacy and security of genetic data. This raises concerns about the privacy of sensitive information since data is stored in external, off-premise data centers. In particular in the health sector, sensitive personal patient records need to be kept confidential [3].

There are a number of technical solutions that have been proposed to protect genome privacy, and existing studies can be categorized into two groups [4]: (i) protecting the

computation process in genome data analysis [5–7] and (ii) protecting the genome data before computation [8, 9] or research outcomes after computation [10].

In order to prevent the user’s genetic data from being compromised by unauthorized users or organizations, protecting genetic privacy is an urgent problem. To mitigate the privacy risks inherent in storing and computing sensitive data, cryptography offers a potential solution in the form of encryption [11]; only the legitimate data owner can access the data by decrypting it using their private decryption key.

However, sometimes the calculation and analysis of genetic data need to be implemented in the cloud due to the limitation of personal computing power and genetic diagnosis algorithm patent, the need for cloud server, the user’s genetic data analysis, and analysis to help get the user diagnosis and treatment of the relevant information. Nevertheless, traditional cryptographic schemes limit the computation process on the ciphertext stored on the cloud and also prevent the data center from performing computation on it without the decryption key.

Homomorphic encryption can do computation on the encrypted data in the case of unknown secret key, and the decryption results of the ciphertext data are equivalent to the corresponding plaintext for the same processing operations. In 2009, Gentry proposed the first FHE scheme and described the framework blueprint of the FHE [12]. Since then many improvements to FHE have been proposed based on Gentry’s work such as [13–17].

Homomorphic encryption-based methods which support secure genome data computation have been studied. Cheon et al. [5] studied how to calculate edit distance of encrypted gene data homomorphically. Yasuda et al. [18] described how to compute multiple Hamming distance values using the LNV scheme [19] on encrypted data. Graepel et al. [20] and Bos et al. [3] applied HE to machine learning and described how to privately conduct predictive analysis based on an encrypted learned model. Lauter et al. [7] gave a solution to privately compute the basic genomic algorithms used in genetic association studies.

To achieve safe genetic data analysis, iDASH (integrating Data for Analysis, Anonymization, and SHaring) National Center has released annual security challenges regarding genetic privacy protection since 2014. In 2016, the challenge of testing for genetic diseases on encrypted genomes (secure outsourcing) was published to calculate the probability of genetic diseases through matching a set of biomarkers to encrypted genomes stored in a commercial cloud service. The requirement is that the entire matching process (only consider the exact match for each variation) needs to be carried out using homomorphic encryption so that no trace is left behind during the computation.

For the challenge published by iDASH, Kim et al. give a practical solution called [KSC17], which uses the homomorphic encryption technique to encrypt the entire gene database as a polynomial on the ring, thus solving the challenge of testing genetic disease (security outsourcing) to a certain extent [21].

The application scenario of this paper is shown in Figure 1. There are three parties involved in this scenario: the user

(hospital or medical institution that has patient’s gene data), the semitrust commercial cloud service, and data owner (the research institute that has the genetic variation database). The purpose of this system is to determine if a patient’s gene data is presented in the gene variation database.

System Initialization. Data owner encrypts the gene variation database and uploads the ciphertexts to the commercial cloud server. Then, the user interacts with the cloud server to complete the testing process. Step 1: the user encrypts the patient’s gene data and uploads the ciphertexts to the commercial cloud server. Step 2: the cloud homomorphically searches user’s gene data in database and generates a ciphertext of searching result. Step 3: the cloud sends the ciphertexts to the user. Step 4: the user decrypts the ciphertexts and concludes whether the patient’s gene data is presented in the gene variation database. The source code of our implementation is available on github <https://github.com/lonyliu/genetest>.

Our Contributions. The contributions of this paper focus on optimizing the design and improving the correctness of the scheme. Through the analysis of the [21] and its related code, we found three types of query errors in [KSC17], called losing of partial coefficient error (LPCE), hash collision error (HCE), and coefficient combination error (CCE), and made some improvements as follows.

- (1) The gene data is encoded by prefix code so as to detect more entries in the gene database with fewer bits than [KSC17].
- (2) Correcting the LPCE error: [KSC17] truncates the variation encodings of gene to 21 bits, which causes partial coefficient losing; thus more than 5% of the entries in the database cannot be queried integrally. In this paper, we decompose the encodings of gene variation by 44 bits, then optimize, encrypt, and query the components, respectively. As a result, all the entries can be queried effectively in the database.
- (3) Correcting the HCE error: [KSC17] uses the method of hash function unreasonably, which may cause HCE error with a probability of 2^{-22} . In this paper, we abandon the hash function by adding half the ciphertext of database, thus avoiding the hash collision.
- (4) Correcting the CCE error: [KSC17] cannot distinguish the different groups of gene data, which may return incorrect results with nonnegligible probability. In this paper, we analyze the relationship between the core parameter l_{snp} (bit size of the encoding for gene variation) and CCE error and specify the condition that parameter l_{snp} needs to satisfy, so that the probability of CCE errors is negligible.

2. Practical Homomorphic Encryptions

This section describes the homomorphic encryption schemes which are used in our genetic privacy protection. First, some symbols and parameters are described below.

For the security parameter λ , let integer $M = M(\lambda)$ define the M th cyclotomic polynomial $\Phi_M(x)$. Throughout

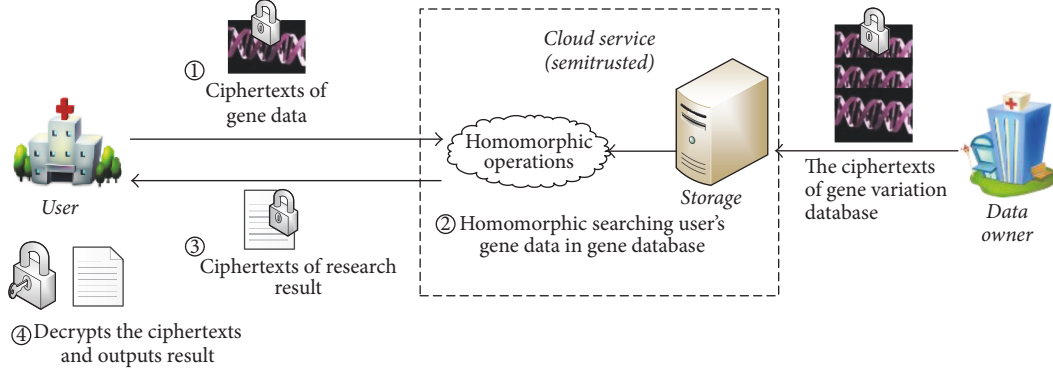


FIGURE 1: The application scenario of secure testing for genetic diseases.

this paper, we assume that the integer M is a power of two so that $N = M/2$ and $\Phi_M(x) = X^N + 1$. Both of our homomorphic encryption schemes operate in the polynomial ring $\mathcal{R} = \mathbb{Z}[x]/\Phi_M(x)$. $[\cdot]_Q$ denotes the reduction modulo Q into the interval $(-Q/2, Q/2] \cap \mathbb{Z}$ of the integer or integer polynomial (coefficient-wise). Set the plaintext space to $\mathcal{R}_t := \mathcal{R}/t\mathcal{R}$ for some fixed $t \geq 2$ and the ciphertext space to $\mathcal{R}_Q := \mathcal{R}/Q\mathcal{R}$ for an integer $Q = Q(\lambda)$. Let $\chi = \chi(\lambda)$ denote a noise distribution over the ring \mathcal{R} . Notation $a \leftarrow D$ denotes that a is chosen from the distribution D , and $a \stackrel{R}{\leftarrow} D$ denote that a is randomly chosen from the distribution D .

We give the brief introduction of the RLWE scheme [22] and the Ring-GSW scheme [23].

2.1. The RLWE Scheme. First basic homomorphic encryption scheme is based on the hardness of Ring Learning with Errors (RLWE) assumption, which is proposed by Lyubashevsky, Peikert, and Regev. The RLWE assumption is divided into decisional RLWE assumption and computational RLWE assumption. The decisional RLWE assumption implies the infeasible solution to distinguish the following two distributions: pairs (a_i, b_i) where $(a_i, b_i) \stackrel{R}{\leftarrow} \mathcal{R}_Q^2$ and pairs $(a_i, b_i) = (a_i, a_i s + e_i)$ where $a_i \stackrel{R}{\leftarrow} \mathcal{R}_Q$ and $s, e_i \stackrel{R}{\leftarrow} \chi$. The computational RLWE assumption is that it is hard to find the key s from many samples $(a_i, b_i) = (a_i, a_i s + e_i)$.

The RLWE scheme is described as follows:

- (i) *RLWE.ParamsGen*(λ): given the security parameter λ , choose an integer $M = M(\lambda)$ which is a power of 2, a ciphertext modulus $Q = Q(\lambda)$, a plaintext modulus t with $t \mid Q$, and discrete Gaussian distribution χ_{err} . Output params $\leftarrow (M, Q, t, \chi_{\text{err}})$.
- (ii) *RLWE.KeyGen*(params): for input parameters M , let $N = \phi(M)$ and choose a random sparse $s \leftarrow \{0, \pm 1\}^N$. Generate an RLWE instance $(a, b) = (a, [-as + e]_Q)$ for $e \leftarrow \chi_{\text{err}}$. Set the secret key $\text{sk} \leftarrow s$ and the public key $\text{pk} \leftarrow (a, b)$.
- (iii) *RLWE.Enc*(m, pk): for the input plaintext $m = \sum_i m_i X^i \in \mathcal{R}_t$, choose a small polynomial $v \in \mathcal{R}$ and

two Gaussian polynomials $e_0, e_1 \leftarrow \mathcal{R}$, and output the ciphertext ct :

$$\text{ct} = (c_0, c_1) = \left(\frac{Q}{t} m + bv + e_0, av + e_1 \right) \in \mathcal{R}_Q^2. \quad (1)$$

- (iv) *RLWE.Dec*(ct, sk): given the ciphertext $\text{ct} = (c_0, c_1)$, output the plaintext m :

$$m \leftarrow \left[\left(\frac{t}{Q} \right) \cdot [c_0 + s \cdot c_1]_Q \right]. \quad (2)$$

- (v) *RLWE.Add*($\text{ct}_1, \text{ct}_2, \text{ct}_3, \text{sk}$): given three ciphertexts $\text{ct}_1, \text{ct}_2, \text{ct}_3$ with the same secret key sk , output the ciphertext $\text{ct} = \text{ct}_1 + \text{ct}_2 + \text{ct}_3 \in \mathcal{R}_Q^2$.

Conversion and Modulus Switching techniques have been introduced in [KSC17]. Conversion technique can change an RLWE ciphertext of $m = \sum_i m_i X^i \in \mathcal{R}_t$ into an LWE encryption of its constant term m_0 . Modulus Switching technique reduces the ciphertext modulus Q down to q while preserving the message, thus reducing the size of ciphertext.

2.2. The Ring-GSW Scheme. In 2013, Gentry et al. proposed an LWE-based homomorphic encryption scheme [16], which uses the approximate eigenvector method to express ciphertext as a matrix, so that the addition and multiplication of ciphertext no longer cause dimension expansion. In this paper, we use its RLWE version introduced by Ducas and Micciancio [23], and its encryption algorithm is given below:

- (i) *RGSW.ParamsGen*(\cdot): given the same parameters and secret key s as in the RLWE scheme, set the decomposition base B_g and exponent d_g satisfying $B_g^{d_g} > Q$. Given a small matrix $\mathbf{G} = (\mathbf{I} \| B_g \mathbf{I} \parallel \dots \parallel B_g^{d_g-1} \mathbf{I}) \in \mathcal{R}_Q^{2d_g \times 2}$ for 2×2 identity matrix \mathbf{I} .
- (ii) *RGSW.Enc*(m, sk): given the plaintext $m \in \mathcal{R}_t$, choose a matrix $\mathbf{a} \stackrel{R}{\leftarrow} \mathcal{R}_Q^{2d_g \times 1}$ uniformly, and $\mathbf{e} \stackrel{R}{\leftarrow} \chi_s$ ($\mathbf{e} \in \mathcal{R}_Q^{2d_g} \simeq \mathbb{Z}^{2d_g \cdot n}$); output the ciphertext CT :

$$\text{CT} = [\mathbf{b}, \mathbf{a}] + m\mathbf{G} \in \mathcal{R}_Q^{2d_g \times 2}, \quad (\mathbf{b} = -\mathbf{a} \cdot \mathbf{s} + \mathbf{e}). \quad (3)$$

TABLE I: The format of genome data.

Chr	Pos	Loc	Ref	Alt	Type
1	160952708	rs2250304	C	T	SNP
1	160952937	rs71090344		GGAGGTTTCAGTGAGCT	INS
1	160953538	rs59471747		T	INS
1	160953667	rs2988723	G	A	SNP
1	160955055	.	TG	CA	SUB
1	160955067	.	GCA	ACG	SUB
1	160955085	.	CTA	TTG	SUB
1	160955294	rs6427571	A	G	SNP
1	160955725	rs2990700	T	C	SNP
1	160956178	rs3007155	T	C	SNP
1	160956420	rs3007156	G	A	SNP
1	160956744	rs2990701	C	T	SNP
1	160957493	rs3007157	A	G	SNP
1	160957862	.		T	INS
1	160957885	rs2483148	A	C	SNP
1	160958160	rs2481072	G	A	SNP
1	160958212	.	C	T	SNP
1	160959961	rs7530765	C	T	SNP
1	160961448	rs11300130	AG		DEL

And the ciphertext CT satisfies $CT \cdot (1, \mathbf{s}) = m \cdot (1, \mathbf{s}, \dots, B_g^{d_g-1}, B_g^{d_g-1} \mathbf{s}) + \mathbf{e}$. Let $\text{Dec}_{B_g}(\cdot)$ denote the decomposition with the base B_g , so m can be regarded as an approximate eigenvalue of $\text{Dec}_{B_g}(CT)$ with the eigenvector $(1, \mathbf{s}, \dots, B_g^{d_g-1}, B_g^{d_g-1} \mathbf{s})$.

Reference [17] defines a hybrid multiplication between an RLWE ciphertext $ct \in \mathcal{R}_Q^2$ and an RGSW ciphertext $CT \in \mathcal{R}_Q^{2d_g \times 2}$.

$$\begin{aligned} ct_{\text{H-mult}} &= \text{Hybrid.Mult}(CT, ct) \\ &= CT^T \cdot \text{Dec}(ct) \in \text{TLWE}(m_{CT} \cdot m_{ct}) \in \mathcal{R}_Q^2. \end{aligned} \quad (4)$$

Thus the ciphertext $ct_{\text{H-mult}}$ is a RLWE encryption of $m_{CT} m_{ct}$.

3. Encoding and Encryption of Gene Data

Recall the task proposed by iDASH: secure biomarkers matching of encrypted genetic data, and in this section, we describe how to encode and encrypt the genomic data.

3.1. Genetic Data. The gene data is stored in a semitrusted business cloud in VCF format. The database VCF file contains multiple genotype information entries, where each of them consists of chrome (chr), position (pos), locus (loc), reference (ref), alternate (alt), type. The example of database is shown in Table I. Chrome represents the chromosome where the gene is located, and it ranges from 1 to 22, X, and Y. Position represents the base position of the gene variation in the chromosome, and locus indicates the location of the gene.

Reference, alternate, type display the base transformation information for the variation: reference represents the base information before the mutation occurs; alternate represents the base information after the mutation; type indicates the type of the mutation, including the single base variation (SNP), multibase mutation (SUB), insertion variation (INS), and deletion variation (DEL).

In fact, the gene mutation can be located by chr and pos information only, and the information of base change can be obtained by comparing the ref base and the alt base. In order to improve the efficiency of the program, we only match the chr and pos information between the patient and the cloud, and then we get the corresponding ref and alt information of base variation at the same location in the database. Finally the user compares the base change information from the cloud and his base change information to get the final match result.

3.2. Encoding and Encryption of Genetic Data. In this section, we describe how to encode the genomic data so that they can be applied to homomorphic encryption scheme. Let d_i denote the position information of the i th entry in the gene database, α_i the variation information of the i th entry in the gene database, α_i^{ref} and α_i^{alt} the integer encodings of reference genome and alternate genome, respectively.

For the coding of the gene position information, define a mapping from (chr, pos) to d_i :

$$\theta : (\mathbb{Z}, \mathbb{Z}) \longrightarrow \mathbb{Z} \quad (5)$$

$$(\text{chrom}, \text{pos}) \longrightarrow d_i = \text{chrom}_i + 24 \cdot \text{pos}_i. \quad (6)$$

In the following we describe how to encode the base variation information in [KSC17]. Firstly, they represent the common SNPs by two binary numbers as

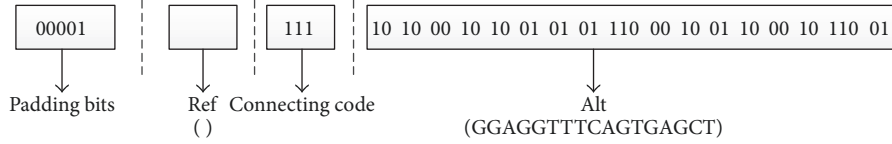


FIGURE 2: Coding structure of base variation information.

$$\begin{aligned}
 A &\rightarrow 00 \\
 T &\rightarrow 01 \\
 G &\rightarrow 10 \\
 C &\rightarrow 11
 \end{aligned} \tag{7}$$

and encode them according to their order. Then pad with 1 to the left of the bit string so as to distinguish the A-string and empty string. For instance, the base A will be encoded as $1 | 00 = (100)_2 = 4$, and string CG will be encoded as $1 | 11 | 10 = (11110)_2 = 30$. n_{SNP} denotes the maximal number of reference (or alternate) alleles to be compared between the query genome and genomes in the target database; thus the length of the base string is $l_{\text{SNP}} = 2 \cdot n_{\text{SNP}} + 1$. And in [KSC17] the encoding of base variation information is expressed as

$$\alpha_i = 2^{l_{\text{SNP}}} \cdot \alpha_i^{\text{ref}} + \alpha_i^{\text{alt}}. \tag{8}$$

Our Contribution. The value of n_{SNP} in [KSC17] is set, respectively, to 2, 5, or 10, but the genovariation that more than 10-base insertion or deletion may occur actually. For example, the second entry in Table 1 for column of “alt” genome is GGAGGTTTCAGT GAGCT. If the patient’s alt genome (query gene information) is GGAGGTTTCA, the server will conclude that the patient is more likely to suffer from a genopathy. At the same time, we found that the numbers of ref bases and alt bases are usually not symmetrical through the statistical analysis of the genetic database, and the number of bases after concatenating the ref and alt genome does not exceed 20 mostly. Therefore, the prefix code is used to encode the genome data:

Firstly, the SNPs are encoded as

$$\begin{aligned}
 A &\rightarrow 00, \\
 T &\rightarrow 01, \\
 G &\rightarrow 10, \\
 C &\rightarrow 110.
 \end{aligned} \tag{9}$$

Then, a string “111” is added to concatenate the ref and alt genome; this can help us to separate the encoding of ref and alt genome correctly. Finally, pad with 1 to the left of the bit string so as to distinguish the A-strings. Here is the formula of getting α_i :

$$\alpha_i = 1 | \alpha_i^{\text{ref}} | 111 | \alpha_i^{\text{alt}}, \quad (\alpha_i \leq 2^{44}). \tag{10}$$

Let l_{SNP} denote the bit size of α_i , and set $l_{\text{SNP}} = 44$ so as to expand the number of gene entries which can be correctly

matched. If the length of α_i is less than 44 bits, then pad bit 0 at the left of the bit string to ensure that the length of the encoding is 44 bits. For the case of length $(\alpha_i) > 44$ bits, α_i is divided by 44 bits; the details are in Section 3.3.3.

For example, the base variation information of the second entry in the Table 1 will be encoded as 0000111101000101001010111000100110001011001. And it can be decoded by the way shown in Figure 2.

After the process of encoding the genetic data, the database file will be encoded as a set of pairs (d_i, α_i) for $i = 1, 2, \dots, n$. The encoding results of Table 1 are shown in Table 2.

The HE scheme in this paper is carried out on a polynomial ring, so it is necessary to express the integer pairs as polynomial $\text{DB}(X) = \sum_{i=0}^N c_i X^i \in \overline{\mathcal{R}}$, where

$$c_i = \begin{cases} \alpha_i, & \text{if } i = d_i, \\ \alpha_i \leftarrow \mathbb{Z}_t & \text{else.} \end{cases} \tag{11}$$

Since the d_i from VCF files have bits size about 32, set $\overline{\mathcal{R}} \triangleq (\mathbb{Z}[X]/[X^{\overline{N}} + 1])$, $\overline{N} = 2^{33} - 1$. And then the data owner (research institute) encrypts the polynomial $\text{DB}(X)$ with the RLWE public-key encryption scheme as described above.

The query genes are also encoded as a pair of integers (d, α) . However, the hospital or medical institution only needs to encrypt the monomial X^{-d} with the RGSW symmetric encryption scheme.

3.3. The Optimization of the Encoded Data. Since the d_i from VCF files have bits size of about 32, set $\overline{\mathcal{R}} \triangleq (\mathbb{Z}[X]/[X^{\overline{N}} + 1])$, $\overline{N} = 2^{33} - 1$. While taking into account the safety and efficiency of HE schemes, a dimension $2^{11} < N < 2^{16}$ is considered appropriate.

[KSC17] makes use of SHA-3 to transform d_i into a pair of two nonnegative integers d_i^* and d_i^\dagger : $\text{Hash}(d_i) \rightarrow (d_i^*, d_i^\dagger)$, and both of them have the bit size of 11 bits; then define ring $\mathcal{R} \triangleq (\mathbb{Z}[X]/[X^N + 1])$, $N = 11$, and mapping $\Phi : \overline{\mathcal{R}}_t \rightarrow \mathcal{R}_t^{k \times 2}$ $\text{DB}(X) \rightarrow (\text{DB}_j^*(X), \text{DB}_j^\dagger(X))_{j \in \{1, \dots, k\}}$, and transform polynomial $\text{DB}(X)$ into k groups of lower-dimension polynomials $\text{DB}_j^*(X) \in \mathcal{R}$, $\text{DB}_j^\dagger(X) \in \mathcal{R}$, where

$$\begin{aligned}
 \text{DB}_j^*(X) &= \sum_{i=0}^N c_{j, d_i^*}^* X^{d_i^*}, \\
 \text{DB}_j^\dagger(X) &= \sum_{i=0}^N c_{j, d_i^\dagger}^\dagger X^{d_i^\dagger},
 \end{aligned} \tag{12}$$

TABLE 2: Encoding of genetic data.

Chr	Pos	d	Ref	Alt	α
1	160952708	3862864993	C	T	477
1	160952937	3862870489		GGAGGTTTCAGTGAGCT	1074435738713
1	160953538	3862884913		T	61
1	160953667	3862888009	G	A	220
1	160955055	3862921321	TG	CA	5880
1	160955067	3862921609	GCA	ACG	222106
1	160955085	3862922041	CTA	TTG	117206
1	160955294	3862927057	A	G	158
1	160955725	3862937401	T	C	382
1	160956178	3862948273	T	C	382
1	160956420	3862954081	G	A	220
1	160956744	3862961857	C	T	477
1	160957493	3862979833	A	G	158
1	160957862	3862988689		T	61
1	160957885	3862989241	A	C	318
1	160958160	3862995841	G	A	220
1	160958212	3862997089	C	T	477
1	160959961	3863039065	C	T	477
1	160961448	3863074753	AG		151

and polynomials DB^*_j and DB^\dagger_j satisfy $\alpha_i = c_{j,d_i^*}^* + c_{j,d_i^\dagger}^\dagger$, for one $j \in \{1, \dots, k\}$.

A corresponding mapping ϕ is defined as the specific mapping from a term in a polynomial $\overline{\mathcal{R}}_t$ to terms in polynomials \mathcal{R}_t^2 .

$$\begin{aligned} \phi : \overline{\mathcal{R}}_t &\longrightarrow \mathcal{R}_t^2 \\ \alpha_i X^{d_i} &\longrightarrow \left(c_{j,d_i^*}^* X^{d_i^*}, c_{j,d_i^\dagger}^\dagger X^{d_i^\dagger} \right), \end{aligned} \quad (13)$$

for one $j \in \{1, \dots, k\}$.

We found that there are three types of errors in [KSC17], named hash collision error (HCE), coefficient combination error (CCE), and losing of partial coefficient error (LPCE). In the following we will describe these errors and our solutions.

3.3.1. Hash Collision Error. KSC17 made use of SHA-3 to transform 33-bit-size d_i into a pair of two 11-bit-size integers d_i^* and d_i^\dagger in order to improve the efficiency of the scheme. The hash function maps 33 bits of information to 22 bits, which may cause the collisions with a probability of 2^{-22} approximately. This collision will result in a searching error. Take 10,000 entries in the database as an example, and suppose that the user queries the position of d_1 , where $\text{hash}(d_1) \rightarrow (d_1^*, d_1^\dagger)$. The probability of at least one hash collision existing between the query and the database, with the same d_1^* and d_1^\dagger , is $1 - (1 - 1/2^{22})^{10000} > 2^{-9}$; that is, the user might get a wrong result with a probability more than 2^{-9} .

What is more, this error cannot be avoided by repeating the algorithm.

Our Contribution. For the HCE error, we abandon the method of hash function, and decompose the index d_i with the basis $N = 2^{11}$, so that d_i can be represented as $d_i = N^2 \cdot d_i^* + N \cdot d_i^\dagger + d_i^\perp$; that is, $d_i = d_i^* \mid d_i^\dagger \mid d_i^\perp$. Then, we extend the mapping Φ to mapping Ψ

$$\begin{aligned} \Psi : \overline{\mathcal{R}}_t &\longrightarrow \mathcal{R}_t^{k \times 3} \\ \text{DB}(X) &\longrightarrow \left(\text{DB}^*_j(X), \text{DB}^\dagger_j(X), \text{DB}^\perp_j(X) \right)_{j \in \{1, \dots, k\}}, \end{aligned} \quad (14)$$

where k is the number of polynomial groups, $\text{DB}^*_j(X) = \sum_{i=0}^n c_{j,d_i^*}^* X^{d_i^*}$, $\text{DB}^\dagger_j(X) = \sum_{i=0}^n c_{j,d_i^\dagger}^\dagger X^{d_i^\dagger}$, $\text{DB}^\perp_j(X) = \sum_{i=0}^n c_{j,d_i^\perp}^\perp X^{d_i^\perp}$, $\alpha_i = c_{j,d_i^*}^* + c_{j,d_i^\dagger}^\dagger + c_{j,d_i^\perp}^\perp$. And extend the corresponding mapping ϕ to mapping ψ

$$\begin{aligned} \psi : \overline{\mathcal{R}}_t &\longrightarrow \mathcal{R}_t^3 \\ \alpha_i X^{d_i} &\longrightarrow \left(c_{j,d_i^*}^* X^{d_i^*}, c_{j,d_i^\dagger}^\dagger X^{d_i^\dagger}, c_{j,d_i^\perp}^\perp X^{d_i^\perp} \right), \end{aligned} \quad (15)$$

for one $j \in \{1, \dots, k\}$.

As a result, we can effectively avoid the collision caused by the compression of the index d_i and solve the HCE problem.

3.3.2. Coefficient Combination Error. In this section, we will describe how the CCE error is happening. For $\forall o, p \in \mathbb{Z}_N$, the

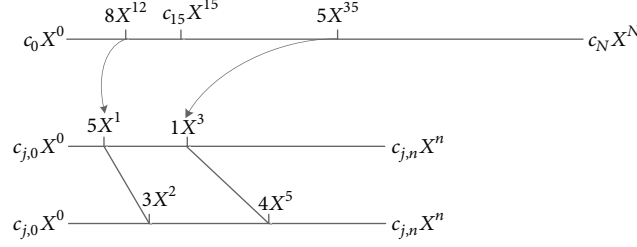


FIGURE 3: A brief description of CCE error.

CCE error exists because [KSC17] cannot distinguish whether two coefficients $c_{j,o}^*$ and $c_{j,p}^\dagger$, picked, respectively, from $(DB_j^*(X), DB_j^\dagger(X))_{j \in \{1, \dots, k\}}$, belong to the one mapping ϕ . This error may lead to the mistake that an entry that is not in the database was judged in the database.

There is a way to determine whether an integer pair (α_i, d_i) is in the database. Firstly the integer pair (α_i, d_i) is employed to represent the polynomial $\alpha_i X^{d_i}$; then transform it into $(c_{d_i}^* X^{d_i}, c_{d_i}^\dagger X^{d_i})$ through mapping ϕ . If there exists a certain group (for one j) $(DB_j^*(X), DB_j^\dagger(X))_{j \in \{1, \dots, k\}}$ whose corresponding entries $c_{j,o}^* X^{d_i}, c_{j,p}^\dagger X^{d_i}$ satisfy $c_{j,o}^* + c_{j,p}^\dagger = \alpha_i$, then the integer pair (α_i, d_i) is judged in the database; otherwise the integer pair (α_i, d_i) is not judged in the database.

We give a brief description in Figure 3.

The first line in Figure 3 represents the polynomial $DB(X)$ with large dimension, and the second and third lines represent the polynomials $DB_j^*(X)$ and $DB_j^\dagger(X)$ with small dimension. All nonzero coefficients in the polynomials are labeled with short lines. If patient want to check whether the query entry $(d_i, \alpha_i) = (15, 9)$ exists in the database with $\text{hash}(15) \rightarrow (1, 5)$, [KSC17] will give the conclusion that this entry exists in the database, since the coefficients of $5X^1$ and $4X^5$ satisfy $5 + 4 = 9$. This will cause the mistake that the patient was misdiagnosed as sick.

Our Contribution. The CCE error means that the query entry (α_i, d_i) does not exist in the database, but the scheme gives the result that the entry is in the database. If the d_i is decomposed into d_i^*, d_i^\dagger , and d_i^\perp , this error will happen only if the sum of the coefficients $c_{j,o}^* + c_{j,p}^\dagger + c_{j,q}^\perp$ of corresponding term $c_{j,o}^* X^{d_i^*}, c_{j,p}^\dagger X^{d_i^\dagger}, c_{j,q}^\perp X^{d_i^\perp}$ for one $j \in \{1, \dots, k\}$ is exactly equal to α_i . Since the sum of coefficients $c_{j,o}^* + c_{j,p}^\dagger + c_{j,q}^\perp \in \mathbb{Z}_t$ is uniformed, the probability of $c_{j,o}^* + c_{j,p}^\dagger + c_{j,q}^\perp = \alpha_i$ for a given α_i is $1/\bar{t} = 1/(2^{l_{\text{snp}}})$, for one group $j \in \{1, \dots, k\}$. The mapping $\Psi: \overline{\mathcal{R}}_t \rightarrow \mathcal{R}_t^{k \times 3}$ will generate k group polynomials, and the probability of at least one group has a collision with the given query entry (α_i, d_i) is $1 - (1 - 1/2^{l_{\text{snp}}})^k$. Through a similar analysis, we find that the probability of CCE error in [KSC17] also satisfies this formula. When [KSC17] gives the parameters $n_{\text{snp}} = 5, l_{\text{snp}} = 11, k = 100$, this probability will be as high as $1 - (1 - 1/2^{l_{\text{snp}}})^k \approx 2^{-4.39} > 2^{-5}$.

There are two factors we need to consider for the parameter l_{snp} . Firstly, the size of coefficients needs to be multiple of 11 bits. Secondly, the size of coefficients needs to be large enough to decrease the probability of CCE error, and the scheme's efficiency should be taken into account. Consequently we set $l_{\text{snp}} = 44$ to reach the requirement for security and efficiency, and decrease the probability of CCE error to $2^{-37.4}$.

3.3.3. Losing of Partial Coefficient Error. In this section, we will describe how the error LPCE is happening. [KSC17] sets $l_{\text{snp}} = 21$ bits, while we found that there are some entries in the database whose encodings are more than 21 bits, and the longest one even needs 272 bits to be represented. For those entries whose encodings are more than 21 bits, [KSC17] truncates the encoding bit to the left 21 bits and abandon other bits. The truncation will cause the LPCE problem. In 3.2 we extend l_{snp} to 44 bits, but also we cannot meet our needs for the correct matching.

Our Contribution. In order to solve this problem, we decompose these long encodings by 44 bits, then optimize, encrypt, and query the components, respectively. Suppose that the given entry with large coefficient is (α_i, d_i) . First, α_i is decomposed by 44 bits; get a set of smaller components $(\alpha_{i,0}, \dots, \alpha_{i,1})$, where $\alpha_i = \alpha_{i,0} \mid \dots \mid \alpha_{i,1}, \alpha_{i,j} \in \mathbb{Z}_t$. Second, we construct and output multiple new entries with smaller coefficients $(\alpha_{i,0}, d_i), \dots, (\alpha_{i,1}, d_i)$. Finally, these entries are mapped by ψ and optimized separately. As a result, we can represent and query all entries effectively in the database and solve the LPCE problem.

4. Secure Searching of Gene Data

Section 4.1 introduces the optimized algorithm for coefficient and dimension of the $DB(X) = \sum_{i=0}^{2^{32}-1} c_i X^i$. Section 4.2 describes the searching algorithm of gene data. Section 4.3 shows our experimental results.

4.1. Optimized Encoding Algorithm of Gene Data

4.1.1. Coefficient Optimization. In order to solve LPCE problem and get an efficient scheme, it is necessary to optimize the coefficients of the polynomial $DB(X)$. Set \bar{t} as the minimum power of 44 bits, which is larger than the bound of α . We found that the length of the encoding α in the database is no

Input: $(\alpha_i, d_i) \in \mathbb{Z}_{\bar{t}} \times \mathbb{Z}_{2^{33}}, 1 \leq i \leq n, m = \lceil \bar{t}/t \rceil$.

- (1) for $i \in \{1, \dots, n\}$ do
- (2) $\alpha_i \in \mathbb{Z}_{\bar{t}} \rightarrow (\alpha_{i,m}, \dots, \alpha_{i,1}) \in \mathbb{Z}_t^m$, where $\alpha_i = t^{m-1} \cdot \alpha_{i,m} + t^{m-2} \cdot \alpha_{i,m-1} + \dots + \alpha_{i,1}$
- (3) set $o = 0$
- (4) for $j \in \{m/4, \dots, 1\}$ do
- (5) if $\alpha_{i,4j} = 0, \alpha_{i,4j-1} = 0, \alpha_{i,4j-2} = 0, \alpha_{i,4j-3} = 0$
- (6) $o = o + 1$
- (7) else break
- (8) output $\{((\alpha_{i,m-4o}, d_i), \dots, (\alpha_{i,m-4o-3}, d_i)) \in (\mathbb{Z}_t \times \mathbb{Z}_{2^{33}})^4, \dots, ((\alpha_{i,4}, d_i), \dots, (\alpha_{i,1}, d_i))\}$

ALGORITHM 1: Encoding genomic data: coefficient optimization.

Input: $(\alpha_{i,l}, d_i) \in \mathbb{Z}_t \times \mathbb{Z}_{2^{33}}, 1 \leq i \leq n, 1 \leq l \leq 4, N = 2^{11}$.

- (1) $d_i \rightarrow (d_i^*, d_i^\dagger, d_i^\perp) \in \mathbb{Z}_t^3, d_i = N^2 \cdot d_i^* + N \cdot d_i^\dagger + d_i^\perp$
- (2) $c_{1,d_i^*}^* \stackrel{R}{\leftarrow} \mathbb{Z}_t, c_{1,d_i^\dagger}^\dagger \stackrel{R}{\leftarrow} \mathbb{Z}_t, c_{1,d_i^\perp}^\perp \leftarrow \alpha_{1,j} - c_{1,d_i^*}^* - c_{1,d_i^\dagger}^\dagger \in \mathbb{Z}_t$
- (3) $\mathfrak{D}_1^* = \{\}, \mathfrak{D}_1^\dagger = \{\}, \mathfrak{D}_1^\perp = \{\}; d_i^* \in \mathfrak{D}_1^*, d_i^\dagger \in \mathfrak{D}_1^\dagger, d_i^\perp \in \mathfrak{D}_1^\perp$
- (4) for $i \in \{2, \dots, n\}$ do
- (5) for $j \in \{1, \dots, k\}$ do
- (6) if $d_i^* \notin \mathfrak{D}_j^*, d_i^\dagger \notin \mathfrak{D}_j^\dagger, d_i^\perp \notin \mathfrak{D}_j^\perp$ then
- (7) $c_{j,d_i^*}^* \stackrel{R}{\leftarrow} \mathbb{Z}_t, c_{j,d_i^\dagger}^\dagger \stackrel{R}{\leftarrow} \mathbb{Z}_t, c_{j,d_i^\perp}^\perp \leftarrow \alpha_{i,l} - c_{j,d_i^*}^* - c_{j,d_i^\dagger}^\dagger \in \mathbb{Z}_t$
- (8) else if $d_i^* \in \mathfrak{D}_j^*, d_i^\dagger \notin \mathfrak{D}_j^\dagger, d_i^\perp \notin \mathfrak{D}_j^\perp$ then
- (9) $c_{j,d_i^\dagger}^\dagger \stackrel{R}{\leftarrow} \mathbb{Z}_t, c_{j,d_i^\perp}^\perp \leftarrow \alpha_{i,l} - c_{j,d_i^*}^* - c_{j,d_i^\dagger}^\dagger \in \mathbb{Z}_t$
- (10) ...
- (11) else if $d_i^* \in \mathfrak{D}_j^*, d_i^\dagger \in \mathfrak{D}_j^\dagger, d_i^\perp \notin \mathfrak{D}_j^\perp$ then
- (12) $c_{j,d_i^\perp}^\perp \leftarrow \alpha_{i,l} - c_{j,d_i^*}^* - c_{j,d_i^\dagger}^\dagger \in \mathbb{Z}_t$
- (13) ...
- (14) else if $d_i^* \in \mathfrak{D}_j^*, d_i^\dagger \in \mathfrak{D}_j^\dagger, d_i^\perp \in \mathfrak{D}_j^\perp$ then
- (15) break
- (16) output $(c_j^*, c_j^\dagger, c_j^\perp) \in \mathcal{R}_t^3, (\mathfrak{D}_j^*, \mathfrak{D}_j^\dagger, \mathfrak{D}_j^\perp)$ where $j \in \{1, \dots, k\}$

ALGORITHM 2: Encoding genomic data: dimension optimization.

more than 272 bits; thus set $\bar{t} = 2^{308}$, which is minimum multiple of 44 bits and larger than 272 bits. We use a general method to decompose $\alpha_i \in \mathbb{Z}_{\bar{t}}$ into $(\alpha_{i,m}, \alpha_{i,m-1}, \dots, \alpha_{i,1}) \in \mathbb{Z}_t^m$ with the basis t , where $\alpha_i = t^{m-1} \cdot \alpha_{i,m} + t^{m-2} \cdot \alpha_{i,m-1} + \dots + \alpha_{i,1}$, $m = \lceil \bar{t}/t \rceil$. In this paper, we set $t = 2^{11}$, and thus $m = 28$. Algorithm 1 presents the coefficient optimization algorithm.

4.1.2. Dimension Optimization. Since the encoded integers d_i from VCF files have bits size of about 32, while taking into account the safety and efficiency for implementation of HE schemes, a dimension $2^{11} < N < 2^{16}$ is considered appropriate. After decomposing the index d_i into d_i^*, d_i^\dagger , and d_i^\perp , where $d_i = d_i^* | d_i^\dagger | d_i^\perp$, we point that if the d_i^*, d_i^\dagger , and d_i^\perp have been decomposed for the previous index, then rebuild a set of polynomials and reassign their corresponding coefficients. Here we set the total groups of $\text{DB}^*(X), \text{DB}^\dagger_j(X), \text{DB}^\perp_j(X)$ to k which means $j \in \{1, \dots, k\}$.

Dimension optimization of encoding algorithm is shown as Algorithm 2.

4.2. Secure Searching Algorithm of Gene Data. This section gives the general framework and complete process of secure searching algorithm, showing the process of our secure testing with details in Figure 4.

4.2.1. Database Encryption. The data owner (research institute) encodes the genomic information as $\text{DB}^*(X), \text{DB}^\dagger(X), \text{DB}^\perp(X)$ and encrypts the polynomials as $(\text{ct}_{\text{DB}}^*, \text{ct}_{\text{DB}}^\dagger, \text{ct}_{\text{DB}}^\perp)$. The process is shown in Algorithm 3. Then the research institute submits the ciphertexts to the commercial cloud service (server).

4.2.2. Query Encryption. The user (hospital or medical institution) encodes the query (α, d) as X^{-d^*}, X^{-d^\dagger} , and X^{-d^\perp} , where $d = d^* | d^\dagger | d^\perp$. Then the user sends the ciphertexts $\text{CT}_Q^*, \text{CT}_Q^\dagger, \text{CT}_Q^\perp$ to the server (commercial cloud service):

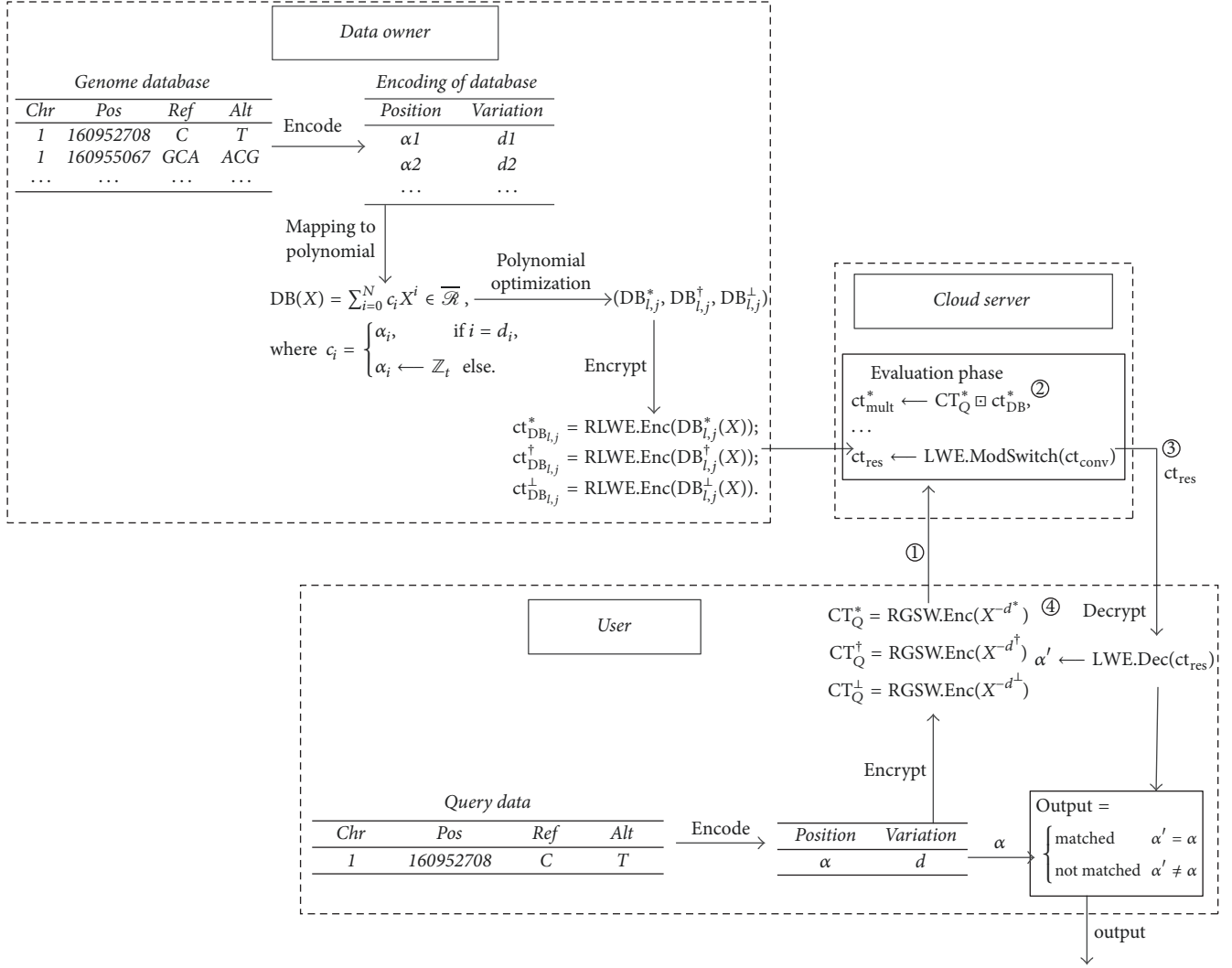
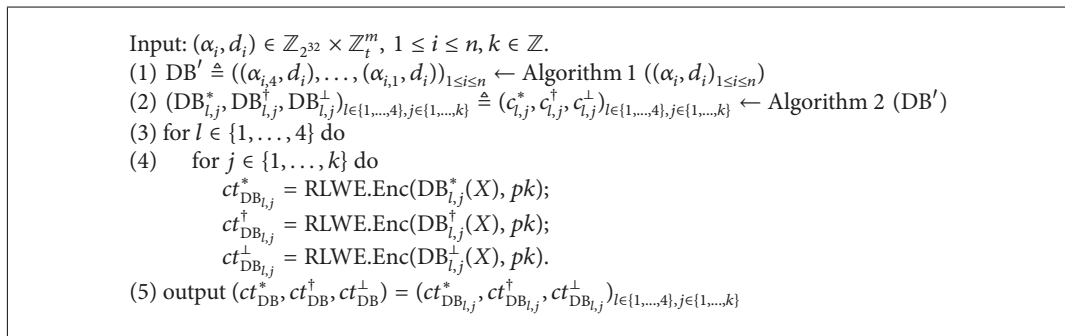


FIGURE 4: The secure testing process for genetic diseases with details.



ALGORITHM 3: Database encoding and encryption.

$$\begin{aligned}
 CT_Q^* &= \text{RGSW.Enc}(X^{-d^*}, pk), \\
 CT_Q^\dagger &= \text{RGSW.Enc}(X^{-d^\dagger}, pk), \\
 CT_Q^\perp &= \text{RGSW.Enc}(X^{-d^\perp}, pk).
 \end{aligned} \tag{16}$$

4.2.3. Evaluation Phase. The server computes the hybrid multiplications ct_{mult}^* , ct_{mult}^\dagger , and ct_{mult}^\perp between the ciphertext of genetic database and the query. Let $ct = ct_{\text{mult}}^* + ct_{\text{mult}}^\dagger + ct_{\text{mult}}^\perp$. The server converts it into an LWE ciphertext and performs Modulus Switching operations. Then return the resulting ciphertext ct_{res} to the user.

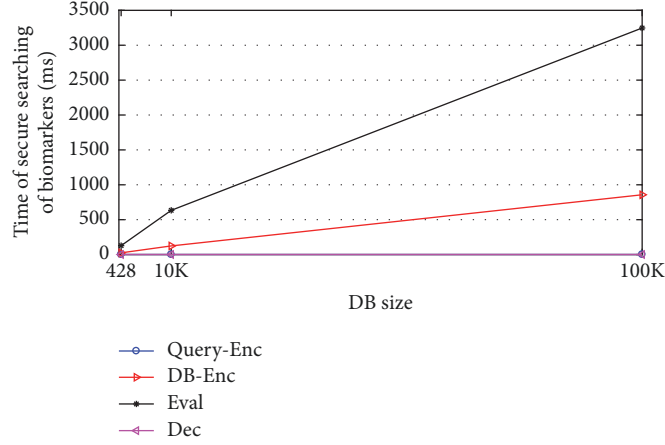


FIGURE 5: Time of secure searching of biomarkers.

$$\begin{aligned}
\text{ct}_{\text{mult}}^* &\leftarrow \text{CT}_Q^* \boxtimes \text{ct}_{\text{DB}}^*, \\
\text{ct}_{\text{mult}}^\dagger &\leftarrow \text{CT}_Q^\dagger \boxtimes \text{ct}_{\text{DB}}^\dagger, \\
\text{ct}_{\text{mult}}^\perp &\leftarrow \text{CT}_Q^\perp \boxtimes \text{ct}_{\text{DB}}^\perp \in \mathcal{R}_Q^2, \\
\text{ct} &= \text{RLWE.Add}(\text{ct}_{\text{DB}}^*, \text{ct}_{\text{DB}}^\dagger, \text{ct}_{\text{DB}}^\perp) \in \mathcal{R}_Q^2, \\
\text{ct}_{\text{conv}} &\in \mathbb{Z}_Q^{N+1} \leftarrow \text{RLWE.Conv}(\text{ct}_{\text{mult}} \in \mathcal{R}_Q^2), \\
\text{ct}_{\text{res}} &\in \mathbb{Z}_q^{N+1} \leftarrow \text{LWE.ModSwitch}(\text{ct}_{\text{conv}} \in \mathbb{Z}_Q^{N+1}).
\end{aligned} \tag{17}$$

4.2.4. *Decryption Phase.* The user decrypts the ciphertext ct_{res} with the secret key and gets the value α' corresponding to the target position d in the database.

$$\alpha' \leftarrow \text{LWE.Dec}(\text{ct}_{\text{res}} \in \mathbb{Z}_q^{N+1}, \text{sk} \in \{-1, 0, 1\}^{N+1}). \tag{18}$$

4.3. *Implementation of Secure Searching.* In fully homomorphic encryption scheme, the homomorphic operations are usually achieved by the polynomial additions and polynomial multiplications as well as bootstrapping processes. As bootstrapping involves costly homomorphic decryption operations, and homomorphic decryption operation requires larger ciphertext modules to prevent decryption errors, this leads to inefficiencies of fully homomorphic encryption. The encryption scheme in this paper is a lattice-based somewhat homomorphic encryption scheme. As few homomorphic operations and no bootstrapping processes are involved in our scheme, a smaller module can ensure the correctness of the scheme. Therefore, the scheme succeeds the efficiency of the [KSC17] scheme. For a 10K database, only hundreds of multiplications and additions of polynomials are needed for the commercial cloud server, so the test results can quickly return. Details are provided in the evaluation phase of Section 4.2.

According to the application scenario of our scheme, we have implemented a three-party (hospital institution, commercial cloud service, and research institute) interactive experimental platform and done experiments on databases

TABLE 3: Time of secure searching of biomarkers (ms).

DB size (number of entries)	Query-Enc	DB-Enc	Eval	Dec
428	22	25	128	1
10K	21	124	634	1
100K	20	857	3247	1

TABLE 4: Storage of secure searching of biomarkers (MB).

DB size (number of entries)	Query	DB	Result
428	160 KB	4	1
10K	160 KB	23	5.6
100K	160 KB	99	24.7

of different sizes (database with 428 entries, 10k entries, 100k entries). We implemented our scheme on a 64-bit single core (i7-6700HQ) at 2.60 GHz, with OS Win 7. The experimental data are listed in Tables 3 and 4, and they are shown in Figures 5 and 6. The source code of our implementation is available on github <https://github.com/lonyliu/genetest>.

Experimental results show that our scheme supports secure searching of gene data for all entries in the genome database (compared to about 5% incorrect searching for gene data in [KSC17]). What is more, based on guaranteeing high efficiency for secure searching of gene data, our scheme reduces the probability of searching error to less than $2^{-37.4}$.

5. Conclusion

In this paper, we discussed how to privately perform secure genomic searching on a semitrusted business cloud with homomorphic encryption. Our scheme can support secure searching of multibase mutation for arbitrary length. What is more, we have solved three errors, hash collision error (HCE),

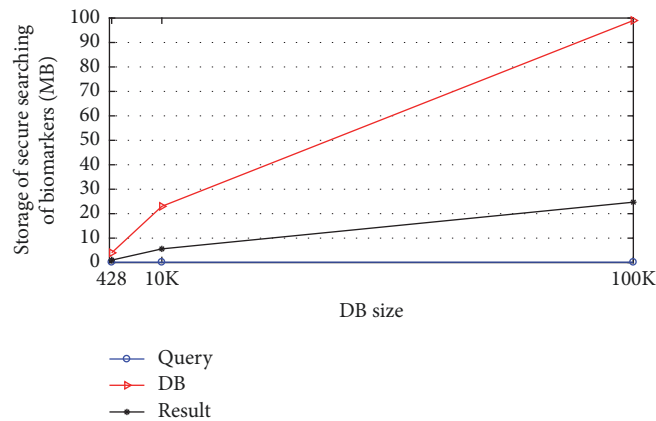


FIGURE 6: Storage of secure searching of biomarkers.

coefficient combination error (CCE), and losing of partial coefficient error (LPCE), in [KSCI17].

Analysis and experimental results show that our scheme can search all the entries in database, compared to about 5% incorrect searching for gene data in [KSCI17], and the probability of searching error is reduced to less than $2^{-37.4}$; thus our scheme can solve the problem of secure searching of gene data based on better homomorphic encryption. We will further study how to eliminate the impact of database capacity on communication traffic in future work.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

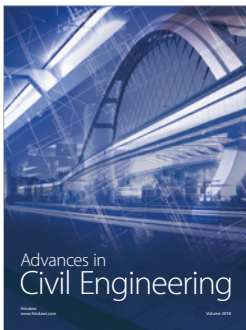
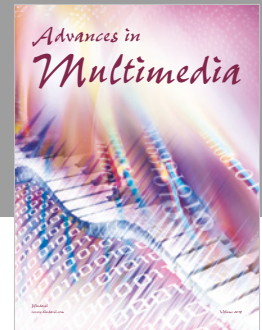
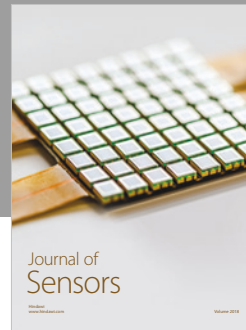
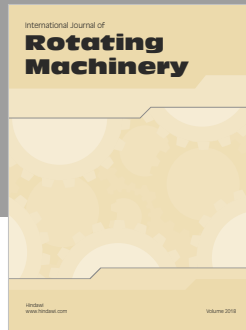
Acknowledgments

This work was supported by National Key R&D Program of China (Grant no. 2017YFB0802000), National Natural Science Foundation of China (Grants nos. U1636114, 61772550, 61572521), National Cryptography Development Fund of China (Grant no. MMJJ20170112), Natural Science Basic Research Plan in Shaanxi Province of China (Grants nos. 2016JQ6037, 2016JQ6030), State Key Laboratory of Information Security (2017-MS-18).

References

- [1] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," *BMC Medical Informatics and Decision Making*, vol. 15, no. 5, article no. S3, 2015.
- [2] H. Tang, X. Jiang, X. Wang et al., "Protecting genomic data analytics in the cloud: state of the art and opportunities," *BMC Medical Genomics*, vol. 9, no. 1, pp. 1–9, 2016.
- [3] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of Biomedical Informatics*, vol. 50, pp. 234–243, 2014.
- [4] S. Wang, Y. Zhang, W. Dai et al., "HEALER: Homomorphic computation of ExAct Logistic rEgReSSion for secure rare disease variants analysis in GWAS," *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2015.
- [5] J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," in *Financial cryptography and data security*, vol. 8976 of *Lecture Notes in Comput. Sci.*, pp. 194–212, Springer, Heidelberg, 2015.
- [6] M. Humbert, E. Ayday, J.-P. Hubaux, and A. Telenti, "Addressing the concerns of the lacks family: Quantification of kin genomic privacy," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*, pp. 1141–1152, deu, November 2013.
- [7] K. Lauter, A. López-Alt, and M. Naehrig, "Private computation on encrypted genomic data," in *Progress in cryptology - LATIN-CRYPT*, pp. 3–27, Springer International Publishing, 2014.
- [8] S. Wang, N. Mohammed, and R. Chen, "Differentially private genome data dissemination through top-down specialization," *BMC Medical Informatics and Decision Making*, vol. 14, no. 1, article no. S2, 2014.
- [9] Y. Zhao, X. Wang, X. Jiang, L. Ohno-Machado, and H. Tang, "Choosing blindly but wisely: Differentially private solicitation of DNA datasets for disease marker discovery," *Journal of the American Medical Informatics Association*, vol. 22, no. 1, pp. 100–108, 2015.
- [10] F. Yu and Z. Ji, "Scalable privacy-preserving data sharing methodology for genome-wide association studies: An application to iDASH healthcare privacy protection challenge," *BMC Medical Informatics and Decision Making*, vol. 14, no. 1, article no. S3, 2014.
- [11] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Manual for Using Homomorphic Encryption for Bioinformatics: This paper provides a new homomorphic encryption algorithm and associated software for bioinformatics to enhance the security and privacy associated with computing on human genomes," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 552–567, 2017.
- [12] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC '09*, pp. 169–178, DBLP, Bethesda, Md, Usa, June 2009.
- [13] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology-EUROCRYPT 2010*, pp. 24–43, Springer Berlin Heidelberg, 2010.
- [14] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE[C] Foundations of Computer Science," *IEEE*, pp. 97–106, 2011.

- [15] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Conference on Innovations in Theoretical Computer Science, ITCS 2012*, pp. 309–325, ACM, New York, NY, USA, January 2012.
- [16] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," *Proceedings of CRYPTO 2013*, vol. 8042, no. 1, pp. 75–92, 2013.
- [17] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds," in *Advances in Cryptology - ASIACRYPT*, pp. 3–33, Springer Berlin Heidelberg, 2016.
- [18] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshihara, "Secure pattern matching using somewhat homomorphic encryption," in *Proceedings of the 2013 ACM Cloud Computing Security Workshop, CCSW 2013 - Co-located with the 20th ACM Conference on Computer and Communications Security, CCS 2013*, pp. 65–76, ACM, Berlin, Germany, November 2013.
- [19] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW '11)*, pp. 113–124, DBLP, Chicago, IL, USA, October 2011.
- [20] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: machine learning on encrypted data," in *Proceedings of the International Conference on Information Security and Cryptology*, vol. 7839, pp. 1–21, Springer-Verlag, 2012.
- [21] M. Kim, Y. Song, and J. H. Cheon, "Secure searching of biomarkers through hybrid homomorphic encryption scheme," *BMC Medical Genomics*, vol. 10, article no. 42, 2017.
- [22] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," in *Advances in cryptology-EUROCRYPT 2010*, vol. 6110, pp. 1–23, Springer Berlin Heidelberg, 2010.
- [23] L. Ducas and D. Micciancio, "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second," in *Proceedings of the Advances in Cryptology - EUROCRYPT*, pp. 617–640, Springer Berlin Heidelberg, 2015.



Hindawi

Submit your manuscripts at
www.hindawi.com

