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

Tan Ping Zhou, Ning Bo Li, Xiao Yuan Yang, Li Qun Lv, Yi Tao Ding, and Xu An Wang

1Key Laboratory of Network & Information Security under the People’s Armed Police, Electronic Department, Engineering University of People’s Armed Police, Xian 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; yxyangyxyang@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 genetic variation) and CCE error and specify the condition that parameter $l_{\text{sup}}$ 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 $\lambda$, let integer $M = M(\lambda)$ define the $M$th cyclotomic polynomial $\Phi_M(x)$. Throughout
The ciphertexts that noise distribution over the ring samples RLWE assumption is that it is hard to find the key.

\[ Q = \frac{M}{2} \]

The decisional RLWE assumption implies the computational RLWE assumption. The RLWE assumption is divided into two Gaussian polynomials \( e_0, e_1 \rightarrow \mathcal{R} \), and output the ciphertext:

\[ \text{ct} = (c_0, c_1) = \left( \frac{1}{Q} m + b v + e_0, a v + e_1 \right) \in \mathcal{R}_Q^2 \]  

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

\[ m \leftarrow \left[ \left( \frac{1}{Q} \right) \cdot [c_0 + s \cdot c_1] \right] \]  

(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 \( \text{sk} \), 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 \( m = \sum m_i X_i \in \mathcal{R}_\gamma \) 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 Duc and Micciancio [23], and its encryption algorithm is given below:

(i) RGSW.ParamsGen(\lambda): given the same parameters and exponents \( B_g \) as in the RLWE scheme, set the decomposition base \( B_g \) and exponent \( d_g \) satisfying \( B_g^d > Q \). Given a small matrix \( \mathbf{G} = (I|B_g^1 I | \cdots | B_g^d I) \in \mathcal{R}_Q^{2d \times 2} \) for \( 2 \times 2 \) identity matrix \( I \).

(ii) RGSW.Enc(\text{m}, \text{pk})\!: given the plaintext \( \text{m} \in \mathcal{R}_\gamma \), choose a matrix \( \mathbf{a} \rightarrow \mathcal{R}_{Q}^{2d \times 1} \) uniformly, and \( \mathbf{e} \rightarrow \mathcal{X}_{\mathbf{e}} \ (\mathbf{e} \in \mathcal{R}_Q^{2d}) \); output the ciphertext CT:

\[ \text{CT} = [\mathbf{b}, \mathbf{a}] + m \mathbf{G} \in \mathcal{R}_Q^{2d \times 2}, \quad (\mathbf{b} = -a \cdot s + e) \]
### Table 1: The format of genome data.

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

And the ciphertext CT satisfies \( CT \cdot (1, s) = m \cdot (1, s, \ldots, B_{d_i}^{d_i-1}, B_{d_i}^{d_i-1}) + e \). Let Dec_{B_{d_i}}(\cdot) denote the decomposition with the base \( B_{d_i} \), so \( m \) can be regarded as an approximate eigenvalue of Dec_{B_{d_i}}(CT) with the eigenvector \( (1, s, \ldots, B_{d_i}^{d_i-1}, B_{d_i}^{d_i-1}) \).

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

\[
ct_{\text{mul}} = \text{Hybrid.Mult}(CT, ct)
\]

\[
= CT^T \cdot \text{Dec}(ct) \in \text{TLWE}(m_{CT} \cdot m_{ct}) \in \mathcal{R}_{CT}^{2}
\]

Thus the ciphertext \( ct_{\text{mul}} \) 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 chromosome (chr), position (pos), locus (loc), reference (ref), alternate (alt), type. The example of database is shown in Table 1. Chromosome 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, \( \alpha_i \) the variation information of the \( i \)th entry in the gene database, \( \alpha_i^{\text{ref}} \) and \( \alpha_i^{\text{alt}} \) 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 \( d_i \):

\[
\theta : (Z, Z) \rightarrow Z
\]

\[
(\text{chr}, \text{pos}) \rightarrow d_i = \text{chrom} + 24 \cdot \text{pos}.
\]
Figure 2: Coding structure of base variation information.

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) and empty string. For instance, the base A will be encoded as 1|00 = (100) and string CG will be encoded as 1|11|10 = (11110). \( n_{SNP} \) 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 \( l_{SNP} = 2 \cdot n_{SNP} + 1 \). And in [KSC17] the encoding of base variation information is expressed as

\[
\alpha_i = 2^{n_{SNP}} \cdot \alpha_i^{ref} + \alpha_i^{alt}.
\]

Our Contribution. The value of \( n_{SNP} \) in [KSC17] is set, respectively, to 2, 5, or 10, but the variation 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. 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

\[
\begin{align*}
A & \rightarrow 00, \\
T & \rightarrow 01, \\
G & \rightarrow 10, \\
C & \rightarrow 110.
\end{align*}
\]

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 \( \alpha_i \):

\[
\alpha_i = 1 \mid \alpha_i^{ref} \mid 111 \mid \alpha_i^{alt}, \quad (\alpha_i \leq 2^{44}).
\]

Let \( l_{SNP} \) denote the bit size of \( \alpha_i \), and set \( l_{SNP} = 44 \) so as to expand the number of gene entries which can be correctly matched. If the length of \( \alpha_i \) 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, \( \alpha_i \) 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 00011111100100010010101100101100100110010011000100101100110001011001100010011000101110001000. 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 \((d_i, \alpha_i)\) for \( i = 1, 2, \ldots, n \). 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 DB(\( X \)) = \( \sum_{i=0}^{\lfloor K/2 \rfloor} c_i X^i \in \mathcal{R} \), where

\[
\begin{align*}
\alpha_i &= \begin{cases} 
\alpha_i, & \text{if } i = d_i, \\
\alpha_i & \text{else} \end{cases} \\
\alpha_i & \leftarrow \mathbb{Z}_q
\end{align*}
\]

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

The query genes are also encoded as a pair of integers \((d, \alpha)\). 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 \( \mathcal{R} \triangleq (\mathbb{Z}[X]/[X^{N} + 1]) \), \( N = 2^{33} - 1 \). While taking into account the safety and efficiency of HE schemes, a dimension \( 2^{31} < N < 2^{36} \) is considered appropriate.

[KSC17] makes use of SHA-3 to transform \( d_i \) into a pair of two nonnegative integers \( d_i^{\ast} \) and \( d_i^{\dagger} \) : Hash(\( d_i \)) \( \rightarrow (d_i^{\ast}, d_i^{\dagger}) \), 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 \( \Phi : \mathcal{R}_{11} \rightarrow \mathcal{R}_{11 \times 2^k} \) DB(\( X \)) \( \rightarrow (DB_{11}^{\dagger}(X), DB_{11}^{\ast}(X))_{i=1,...,k} \) and transform polynomial DB(\( X \)) into \( k \) groups of lower-dimension polynomials \( DB_{11}^{\dagger}(X) \in \mathcal{R}, DB_{11}^{\dagger}(X) \in \mathcal{R} \), where

\[
\begin{align*}
DB_{11}^{\dagger}(X) &= \sum_{i=0}^{N} c^{\dagger}_i X^{d_i^{\dagger}}, \\
DB_{11}^{\ast}(X) &= \sum_{i=0}^{N} c^*_i X^{d_i^{\ast}}.
\end{align*}
\]
and polynomials $\text{DB}^*_j$ and $\text{DB}^\perp_j$ satisfy $\alpha_i = c_{j,d^*_i} + c_{j,d^\perp_i}$ for one $j \in \{1, \ldots, k\}$.

A corresponding mapping $\phi$ is defined as the specific mapping from a term in a polynomial $\bar{R}_i$ to terms in polynomials $\bar{R}_i^2$.

$$
\phi : \bar{R}_i \rightarrow \bar{R}_i^2$
$$
$$
\alpha_i X^{d_i} \rightarrow \left(c^*_i, X^{d^*_i}, c^\perp_i, X^{d^\perp_i}\right),
$$
(13)
for one $j \in \{1, \ldots, k\}$.

We found that there are three types of errors in [KSCI7], 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. KSCI7 made use of SHA-3 to transform 33-bit-size $d_i$ into a pair of two 11-bit-size integers $d_i^*$ and $d_i^\perp$ 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 $d_i$, where $\text{hash}(d_i) \rightarrow \{d_i^*, d_i^\perp\}$. The probability of at least one hash collision existing between the query and the database, with the same $d_i^*$ and $d_i^\perp$, 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 $d_i = N^2 \cdot d_i^* + N \cdot d_i^\perp + d_i^\perp$; that is, $d_i = d_i^* | d_i^\perp | d_i^\perp$. Then, we extend the mapping $\Phi$ to mapping $\Psi$

$$
\Pi : \bar{R}_i \rightarrow \bar{R}_i^{2k^3}
$$
(14)
$$
\text{DB}(X)
$$
$$
\rightarrow \left(\text{DB}^*_j(X), \text{DB}^\perp_j(X), \text{DB}^\perp_j(X)\right)_{j \in \{1, \ldots, k\}},
$$
where $k$ is the number of polynomial groups, $\text{DB}^*_j(X) = \sum_{i=0}^n c_{j,d^*_i} X^{d^*_i}$, $\text{DB}^\perp_j(X) = \sum_{i=0}^n c_{j,d^\perp_i} X^{d^\perp_i}$, and $\text{DB}^\perp_j(X) = \sum_{i=0}^n c_{j,d^\perp_i} X^{d^\perp_i}$.

And extend the corresponding mapping $\phi$ to mapping $\psi$

$$
\psi : \bar{R}_i \rightarrow \bar{R}_i^{2^3}
$$
(15)
$$
\alpha_i X^{d_i} \rightarrow \left(c^*_i, X^{d^*_i}, c^\perp_i, X^{d^\perp_i}, c^\perp_i, X^{d^\perp_i}\right),
$$
for one $j \in \{1, \ldots, k\}$.

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

<table>
<thead>
<tr>
<th>Chr</th>
<th>Pos</th>
<th>Ref</th>
<th>Alt</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>160952708</td>
<td>3862864993</td>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>160952937</td>
<td>3862870489</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>160953358</td>
<td>3862884193</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>160953667</td>
<td>3862893501</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160955024</td>
<td>3862912157</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>160955085</td>
<td>3862912204</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160955294</td>
<td>3862912705</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160955725</td>
<td>3862913740</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>160956178</td>
<td>3862914827</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>160956420</td>
<td>3862914827</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>160956744</td>
<td>3862916857</td>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>160957743</td>
<td>3862917983</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160957862</td>
<td>3862918868</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160957885</td>
<td>3862919241</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160958160</td>
<td>3862919584</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>160958212</td>
<td>3862919708</td>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>160959991</td>
<td>3863039065</td>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>160961448</td>
<td>3863074753</td>
<td>AG</td>
<td></td>
</tr>
</tbody>
</table>
CCE error exists because [KSC17] cannot distinguish whether two coefficients \( c_{i,o}^* \) and \( c_{j,p}^\perp \), picked, respectively, from \((DB_1^s(X), DB_1^b(X))_{i\in[1,...,k]}\), belong to the one mapping \( \phi \). 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 \((\alpha_i, d_i)\) is in the database. Firstly the integer pair \((\alpha_i, d_i)\) is judged in the database; otherwise the integer pair \((\alpha_i, d_i)\) 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 \( DB^s(X) \) and \( DB^b(X) \) 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 hash(15) \( \rightarrow (1,5)\), [KSC17] will give the conclusion that this entry exists in the database, since the coefficients of 5\(X^1\) and 4\(X^5\) 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 \((\alpha_i, d_i)\) 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 \( d^a_i, d^b_i, \) and \( d^c_i \), this error will happen only if the sum of the coefficients \( c_{i,o}^* + c_{j,p}^\perp \) of corresponding term \( c_{j,p}^\perp X^{d^c_i} \), \( c_{j,p}^\perp X^{d^c_i} \), \( c_{j,p}^\perp X^{d^c_i} \) for one \( j \in \{1, \ldots, k\} \) is exactly equal to \( \alpha_i \). Since the sum of coefficients \( c_{i,o}^* + c_{j,p}^\perp \in Z^k \) is uniformed, the probability of \( c_{i,o}^* + c_{j,p}^\perp = \alpha_i \) for a given \( \alpha_i \) is 1/\( 2^{21} \), for one group \( j \in \{1, \ldots, k\} \). The mapping \( \Psi : \mathbb{R}^{k^2} \rightarrow \mathbb{R}^{k^2} \) will generate \( k \) group polynomials, and the probability of at least one group has a collision with the given query entry \((\alpha_i, d_i)\) is \( 1 - (1 - 1/2^{21})^k \). Through a similar analysis, we find that the probability of CCE error in [KSC17] also satisfies this formula. When [KSC17] gives the parameters \( n_{\text{sup}} = 5, l_{\text{sup}} = 11, k = 100 \), this probability will be as high as 1 - \((1 - 1/2^{21})^k \) \( \approx 2^{-4.39} > 2^{-5} \).

There are two factors we need to consider for the parameter \( l_{\text{sup}} \). 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 \( l_{\text{sup}} = 44 \) 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 \( l_{\text{sup}} = 21 \) 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 \( l_{\text{sup}} \) 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 \((\alpha_i, d_i)\). First, \( \alpha_i \) is decomposed by 44 bits; get a set of smaller components \((\alpha_i, d_i, \ldots, \alpha_1)\), where \( \alpha_i = \alpha_i, \alpha_1 = \alpha_i, \alpha_j \in Z^k \). Second, we construct and output multiple new entries with smaller coefficients \((\alpha_i, d_i, \ldots, \alpha_1, d_i)\). Finally, these entries are mapped by \( \psi \) 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) = \( \sum_{i=0}^{2^3-1} c_i X^i \). 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 \( f \) as the minimum power of 44 bits, which is larger than the bound of \( \alpha \). We found that the length of the encoding \( \alpha \) in the database is no...
more than 272 bits; thus set \( t = 2^{108} \), which is minimum multiple of 44 bits and larger than 272 bits. We use a general method to decompose \( \alpha_i \in \mathbb{Z}_T \) into \((\alpha_{i,m}, \alpha_{i,m-1}, \ldots, \alpha_{i,1}) \in \mathbb{Z}^m \) with the basis \( t \), where \( \alpha_i = t^{m-1} \cdot \alpha_{i,m} + t^{m-2} \cdot \alpha_{i,m-1} + \cdots + \alpha_{i,1} \). 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 \( 2^{11} < N < 2^{64} \) is considered appropriate. After decomposing the index \( d_i \) into \( d_i^*, d_i^j, d_i^r, \) and \( d_i^t, \) where \( d_i = d_i^* \mid d_i^j \mid d_i^r \mid d_i^t \), we appoint that if the \( d_i^*, d_i^j, \) and \( d_i^r \) 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 \( \text{DB}^r_j(X), \text{DB}^j(X), \text{DB}^t_j(X) \) to \( k \) which means \( j \in \{1, \ldots, k\} \).

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 \( \text{DB}^r(X), \text{DB}^j(X) \), \( \text{DB}^t(X) \) and encrypts the polynomials as \( (ct_{\text{DB}}^*, ct_{\text{DB}}^j, ct_{\text{DB}}^r) \). 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 institute) encodes the query \((\alpha, d)\) as \( X^{-d^*} \), \( X^{-d^j} \), and \( X^{-d^r} \), where \( d = d^* \mid d^j \mid d^r \). Then the user sends the ciphertexts \( CT^r, CT^j, CT^t \) to the server (commercial cloud service)
Input: \((\alpha_i, d_i) \in \mathbb{Z}_2 \times \mathbb{Z}_m, 1 \leq i \leq n, k \in \mathbb{Z}\).

1. \(DB^i \triangleq ((\alpha_{i,1}, d_{i,1}), \ldots, (\alpha_{i,n}, d_{i,n})) \leftarrow \text{Algorithm 1} ((\alpha_i, d_i) \in \mathbb{Z}_2 \times \mathbb{Z}_m)\)
2. \((DB^i_1, DB^i_2, DB^i_3)_{\ell \in \{1, \ldots, i\}, j \in \{1, \ldots, k\}} \leftarrow \text{Algorithm 2} (DB^i)\)
3. for \(l \in \{1, \ldots, 4\}\) do
   4. for \(j \in \{1, \ldots, k\}\) do
      5. \(ct^*_{DB^i_l,j} = \text{RLWE.Enc}(DB^i_{l,j}(X)), pk)\);
      6. \(ct^t_{DB^i_l,j} = \text{RLWE.Enc}(DB^i_l(X), pk)\);
      7. \(ct^\bot_{DB^i_l,j} = \text{RLWE.Enc}(DB^i_{\bot,l,j}(X), pk)\).
3. output \((ct^*_{DB^i_1}, ct^t_{DB^i_2}, ct^\bot_{DB^i_3}) = (ct^*_{DB^i_{1,j}}, ct^t_{DB^i_{2,j}}, ct^\bot_{DB^i_{3,j}})_{\ell \in \{1, \ldots, i\}, j \in \{1, \ldots, k\}}\)

Algorithm 3: Database encoding and encryption.

\[
\begin{align*}
CT^*_Q &= \text{RGSW.Enc} \left( X^{-d^*}, pk \right), \\
CT^t_Q &= \text{RGSW.Enc} \left( X^{-d^t}, pk \right), \\
CT^\bot_Q &= \text{RGSW.Enc} \left( X^{-d^\bot}, pk \right).
\end{align*}
\]

4.2.3. Evaluation Phase. The server computes the hybrid multiplications \(ct^*_{\text{mult}}, ct^t_{\text{mult}}, \) and \(ct^\bot_{\text{mult}}\) between the ciphertext of genetic database and the query. Let \(ct = ct^*_{\text{mult}} + ct^t_{\text{mult}} + ct^\bot_{\text{mult}}\). The server converts it into an LWE ciphertext and performs Modulus Switching operations. Then return the resulting ciphertext \(ct_{\text{res}}\) to the user.
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 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),
coefficient combination error (CCE), and losing of partial coefficient error (LPCE), in [KSC17].

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 [KSC17], 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


