TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/18 pp333–343
DOI: 10.26599/TST.2021.9010006
Volume 27, Number 2, April 2022

Secure Scheme for Locating Disease-Causing Genes Based on Multi-Key Homomorphic Encryption

Tanping Zhou*, Wenchao Liu, Ningbo Li, Xiaoyuan Yang*, Yiliang Han, and Shangwen Zheng

Abstract: Genes have great significance for the prevention and treatment of some diseases. A vital consideration is the need to find a way to locate pathogenic genes by analyzing the genetic data obtained from different medical institutions while protecting the privacy of patients' genetic data. In this paper, we present a secure scheme for locating disease-causing genes based on Multi-Key Homomorphic Encryption (MKHE), which reduces the risk of leaking genetic data. First, we combine MKHE with a frequency-based pathogenic gene location function. The medical institutions use MKHE to encrypt their genetic data. The cloud then homomorphically evaluates specific gene-locating circuits on the encrypted genetic data. Second, whereas most location circuits are designed only for locating monogenic diseases, we propose two location circuits (TH-intersection and Top-q) that can locate the disease-causing genes of polygenic diseases. Third, we construct a directed decryption protocol in which the users involved in the homomorphic evaluation can appoint a target user who can obtain the final decryption result. Our experimental results show that compared to the JWB+17 scheme published in the journal *Science*, our scheme can be used to diagnose polygenic diseases, and the participants only need to upload their encrypted genetic data once, which reduces the communication traffic by a few hundred-fold.

Key words: public key encryption; Multi-Key Homomorphic Encryption (MKHE); fully homomorphic encryption; disease-causing genes; secure location of disease-causing genes

1 Introduction

The gene is the code that guides human activities, and almost all human life and physiological phenomena are directly related to genes. Genetic data can be widely used in healthcare, biomedical research[1, 2] and identification, and the prevention and treatment of diseases, such as cancer and albinism. Such diseases are generally caused by a malignant mutation in the genes, so identifying the location of these pathogenic genes via genetic diagnosis is an important prerequisite for targeted disease treatment. As such, the development of a scheme for locating disease-causing genes is of vital importance.

With the continuous developments in biomedicine and the decreasing cost of genome sequencing[3], more and more people can access their own genetic data. To identify the root cause of a patient's disease, the patient's genome must be compared with as many other genomes as possible, both those affected and unaffected, related and unrelated[4], to increase the accuracy of locating disease-causing genes. However, personalized genomes

• Tanping Zhou is with the College of Cryptography Engineering, Engineering University of People's Armed Police, Xi'an 710086, China, and with TCA Laboratory, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China. E-mail: tanping2020@iscas.ac.cn.
• Wenchao Liu, Xiaoyuan Yang, Yiliang Han, and Shangwen Zheng are with the College of Cryptography Engineering, Engineering University of People's Armed Police, Xi'an 710086, China. E-mail: liuwch3@mail2.sysu.edu.cn; yxyangyxyang@163.com; hanyil@163.com; 2237246329@qq.com.
• Ningbo Li is with the People's Armed Police Command College China, Tianjin 300100, China. E-mail: 372726936@qq.com.
∗ To whom correspondence should be addressed.
Manuscript received: 2020-09-14; revised: 2020-12-17; accepted: 2021-01-25

contain characteristics that have strong personal privacy implications[5], and if insufficiently protected, it may lead to problems such as genetic discrimination[6] and criminal activity. Therefore, resolving the contradiction between the protection of privacy and the effective analysis of genetic data is an urgent problem in the healthcare industry. In 2017, Jagadeesh et al.[4] provided an alternative scheme JWB+17 for secure location of disease-causing gene, which combines a cryptographic method called Yao's protocol[7] with frequency-based clinical genetics for diagnosing causal disease mutations in patients with monogenic disorders. As reported in Ref. [4], three Boolean circuits are useful for patient diagnosis: the intersection, MAX, and SET DIFF circuits, and Ref. [4] proposed a "two-cloud" model to extend Yao's protocol and realized a secure multiparty computation protocol. However, in our analysis of JWB+17, we found the following defects:

(1) The Boolean circuits used in Ref. [4] are only applicable to monogenic diseases and cannot be applied to polygenic diseases, and the construction of the SET DIFF circuit is not sufficiently compact, which affects the circuit's implementation efficiency.

(2) To circumvent the limitation that requires all participating parties to be online during the execution of Yao's protocol and have the potential for a communication breakdown, Ref. [4] used a "two-cloud" model for general multiparty computation. However, this model is based on the assumption that the two-cloud servers are non-colluding, which is not always the case in the real world, and may represent a large security risk. In addition, the two-cloud model requires more complex gabled circuits than the original Yao's protocol, which affects the scheme's efficiency. Moreover, these gabled circuits cannot be reused during protocol execution, which results in heavy communication traffic.

To address the above two issues, in this paper, we consider the use of Multi-Key Homomorphic Encryption (MKHE)[8] to enable the secure computation and analysis of genetic data. With MKHE, arbitrary operations on encrypted data can be performed under different public keys (participants), which means it is more powerful than the traditional Fully Homomorphic Encryption (FHE)[9, 10], and the final ciphertext can be jointly decrypted by all the participants[11]. MKHE is a powerful tool for realizing secure multiparty computation[12], in which every participant can protect their own genetic data by holding their own secret keys, with no need for the cloud to be non-colluding. The encryption of the same genetic data can also be reused, which provides a better solution for securely locating a disease-causing gene. At present, the security of most MKHE schemes can be reduced to the worst-case hardness of ideal lattice problems. References [13–16] also attempt to analyze gene data using homomorphic encryption schemes, but their schemes are based on single-key homomorphic encryption and cannot be applied to scenarios with multi-source gene data.

Our work: In this paper, we propose a secure scheme for locating disease-causing genes based on MKHE. Our scheme can locate pathogenic genes by homomorphically analyzing the ciphertexts of multiple different medical institutions, thus significantly reducing the risk of leaking genetic data.

(1) As MKHE supports computations for encrypted data from different participants, we combine MKHE and a frequency-based pathogenic gene location function. The medical institutions encrypt their genetic data using MKHE scheme CCS19[17], then the cloud homomorphically implements the specific location circuits, and the resulting ciphertext is sent to all the involved medical institutions for joint decryption.

(2) We propose two location circuits Threshold (TH)-intersection & Top-q), which can be used for the diagnosis of polygenic diseases. Most current location circuits are only suitable for monogenic disease, as they set a fixed threshold, whereas the TH-intersection circuit outputs all the gene positions, for which the sum of the mutated genetic data of all patients is larger than a given threshold. By setting a fixed parameter q, the Top-q circuit outputs the Top-q gene positions that exhibit a high frequency of common mutation among all patients.

(3) We construct a directed decryption protocol that allows the evaluated ciphertext to be decrypted by any target user. Note that the evaluated ciphertext in CCS19 and the most current MKHE schemes are decrypted by all involved users, each of whom will also receive the results, which may be inappropriate for a scenario, in which the evaluated ciphertext should be decrypted by only appointed (target) user. For this situation, we construct a directed decryption protocol, in which the users involved in the homomorphic evaluation can appoint a target user, who will receive the final decryption results, thereby enhancing the ability of the data owner to control their messages.

Experimental results show that compared to the JWB+17 scheme, the participants in our scheme must only upload their encrypted genetic data once, and the

communication traffic is reduced by a few hundred-fold. In addition, our scheme does not require the participants to be online in real time, which makes it more suitable for systems with limited bandwidth.

2 Preliminary

2.1 Definition of MKHE

Definition 1 Leveled MKHE[9]. Given the security parameter λ, let $\mathcal{C}$ be a class of circuits. A leveled multi-key FHE scheme $\mathcal{E} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Extend}, \text{Eval}, \text{Dec})$ is described as follows:

- $pp \leftarrow \mathcal{E}.\text{Setup}(1^\lambda, 1^K, 1^L)$: Given the security parameter λ, the circuit depth L, and the upper bound K on the number of users, output the public parameters pp.
- $(pk_i, sk_i, evk_i) \leftarrow \mathcal{E}.\text{KeyGen}(pp)$: Given the public parameter pp, derive and output of the public key pk_i, secret key sk_i, and the evaluation keys evk_i of user $i (i = 1, \ldots, K)$.
- $c_i \leftarrow \mathcal{E}.\text{Enc}(pk_i, m_i)$: Given the public key pk_i and message m_i, output a ciphertext c_i.
- $\hat{c}_{S_i} \leftarrow \mathcal{E}.\text{Extend}(c_i, (pk_{i_1}, \ldots, pk_{i_k}), em_i)$: On the input of a ciphertext c_i, the public keys $(pk_{i_1}, \ldots, pk_{i_k})$ corresponding to user set S_i, which are incorporated into the homomorphic computations for participant i, and the related materials em_i used in the ciphertext extension, output the extended ciphertext $\hat{c}_{S_i}$, whose corresponding secret key is a combination of $(sk_{i_1}, \ldots, sk_{i_k})$.
- $\hat{c}_S \leftarrow \mathcal{E}.\text{Eval}(\mathcal{C}, (\hat{c}_{S_1}, pk_{S_1}, evk_{S_1}), \ldots, (\hat{c}_{S_t}, pk_{S_t}, evk_{S_t}))$: Upon the input of a Boolean circuit $\mathcal{C}$ along with t tuples $(\hat{c}_{S_i}, pk_{S_i}, evk_{S_i})_{i=[1,t]}$, with each tuple comprising a ciphertext $\hat{c}_{S_i}$ corresponding to user set S_i, a set of public keys $pk_{S_i} = \{pk_j, \forall j \in S_i\}$, and the evaluation keys evk_{S_i}, output the evaluated ciphertext $\hat{c}_S$, corresponding to user set $S = \cup_{i=1}^{t} S_i \subseteq [1, K]$.
- $m_S \leftarrow \mathcal{E}.\text{Dec}((sk_{i_1}, sk_{i_2}, \ldots, sk_{i_k}), c_S)$: Given a ciphertext c_S corresponding to a set of users $S = \{i_1, i_2, \ldots, i_k\} \subseteq [1, K]$, and secret keys $sk_S = \{sk_{i_1}, sk_{i_2}, \ldots, sk_{i_k}\}$, output the message m_S.

Definition 2 Correctness of MKHE. Upon input of any circuit $\mathcal{C}$ of depth at most L, a set of tuples$\{(c_{S_i}, pk_{S_i}, evk_{S_i})\}_{i\in[1,t]}$, and secret keys $sk_{S_i} = \{sk_j, \forall j \in S_i\}$, let $\mu_i = \text{Dec}(sk_{S_i}, c_{S_i})$, and $S = \cup_{i=1}^{t} S_i \subseteq [1, K]$. A leveled MKHE scheme $\mathcal{E}$ is correct if it holds that

$$\text{Dec}(sk_S, \text{Eval}(\mathcal{C}, (c_{S_i}, pk_{S_i}, evk_{S_i})_{i\in[1,t]})) = \mathcal{C}(\mu_1, \ldots, \mu_t).$$

Definition 3 Compactness of MKHE. A leveled MKHE scheme is compact if there exists a polynomial $poly(\cdot, \cdot, \cdot)$, such that $|c| \leqslant poly(\lambda, K, L)$, which means that the length of c is independent of the circuit $\mathcal{C}$, but depends on the security parameter λ, the number of users K, and the circuit depth L.

Preprocessing of genetic data: The research subjects go to a medical institution to have their genes sequenced and to obtain their own genetic data. The medical institution creates a mutated vector for every research subject, and compares the obtained genetic data with those in the genetic mutation database (the items in the database are the positions and mutation information of the genetic mutation). If the researcher's mutation matches that in the database at a specific gene position, then the corresponding gene position vector is set to 1, otherwise it is set to 0. As such, each research subject has a unique bit string for their own genetic variation information. Figure 1 shows a flowchart of the preprocessing operation.

2.2 CCS19 scheme

The underlying MKHE scheme is based on an optimized CCS19 scheme (see Section 3.2 for the optimization of CCS19), which is based on underlying Learning With Errors (LWE) and Ring GSW (RGSW) schemes. We briefly introduce these schemes below. Interested readers

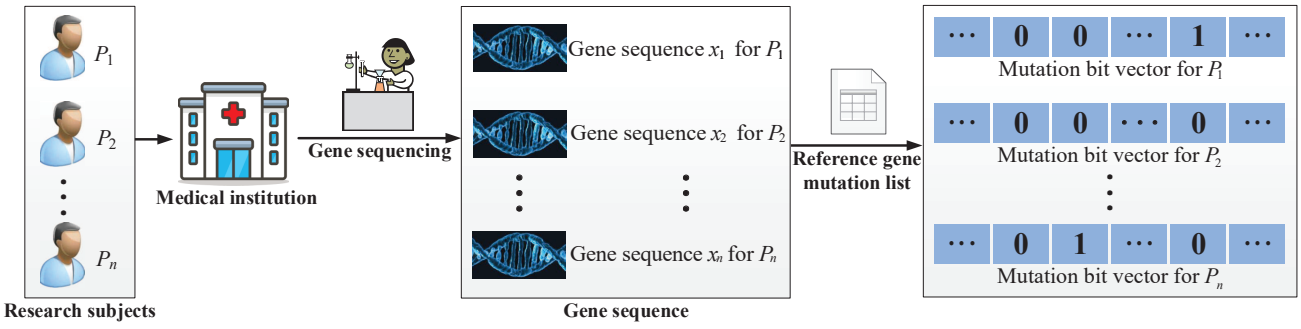

Fig. 1 Schematic of genetic data preprocessing.

can access further details in Ref. [17].

2.2.1 LWE scheme

LWE. Setup$(1^\lambda)\to(pp^{\mathrm{LWE}})$: Input the security parameter λ, output public parameter pp^{LWE}.

LWE.Enc$(m,s)\to(\boldsymbol{ct})$: Input message $m\in\{0,1\}$ and secret $\boldsymbol{s}$, output ciphertext $\boldsymbol{ct}=(b,\boldsymbol{a})\in\mathcal{T}^{n'+1}$, where real Torus $\mathcal{T}=\mathbf{R}\bmod 1$, n' is the dimension of $\boldsymbol{a}$.

We describe here a symmetric encryption for simplicity, but this algorithm can be replaced by any LWE-style encryption schemes, such as public key encryption[18]. The requirement is that the output ciphertext must be a vector $\boldsymbol{ct}=(b,\boldsymbol{a})\in\mathcal{T}^{n'+1}$ satisfying $b+\langle\boldsymbol{a},\boldsymbol{s}\rangle\approx m(\bmod 1)/4$.

LWE.KSGen$(\boldsymbol{t},\boldsymbol{s})\to(KS=\{\boldsymbol{KS}_j\}_{j\in[1,n'']}\in(\mathcal{T}^{d_{ks}\times(n'+1)})^{n''})$: Input LWE secrets $\boldsymbol{t}\in\mathbf{Z}^{1,n''}$ before switching, and $\boldsymbol{s}\in\mathbf{Z}^{n'}$ after switching, output key-switching key $KS=\{\boldsymbol{KS}_j\}_{j\in[1,n'']}\in(\mathcal{T}^{d_{ks}\times(n'+1)})^{n''}$, where n'' and n' are the dimensions of LWE sample, and d_{ks} is the decomposing parameter.

We can transform an LWE ciphertext corresponding to $\boldsymbol{t}$ into another LWE encryption of the same message under the secret $\boldsymbol{s}$ using a key-switching key LWE.KSGen$(\boldsymbol{t},\boldsymbol{s})\to(KS=\{\boldsymbol{KS}_j\}_{j\in[1,n'']}\in(\mathcal{T}^{d_{ks}\times(n'+1)})^{n''})$.

LWE.MKSwitch$(\overline{ct},\overline{KS})\to(\overline{ct'})$: Input extended ciphertext $\overline{ct}=(b,\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k)\in\mathcal{T}^{kn''+1}$, and key-switching keys $\overline{KS}=\{KS_i=\{\boldsymbol{KS}_{i,j}\}_{j\in[1,n'']}\}_{i\in[1,k]}$, output $\overline{ct'}=(b',\boldsymbol{a}'_1,\ldots,\boldsymbol{a}'_k)\in\mathcal{T}^{kn''+1}$, where k is the number of the users involved in the key-switching process.

Here, we consider the notion of the extended LWE encryption and the multi-keyswitching procedure. For k LWE secrets $\boldsymbol{s}_1,\ldots,\boldsymbol{s}_k\in\mathbf{Z}^{n'}$, an extended ciphertext $\overline{ct}=(b,\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k)\in\mathcal{T}^{kn''+1}$ is called an encryption of $m\in\{0,1\}$ with respect to the concatenated secret $\bar{\boldsymbol{s}}=(\boldsymbol{s}_1,\ldots,\boldsymbol{s}_k)$ if $b+\sum_{i=1}^{k}\langle\boldsymbol{a}_i,\boldsymbol{s}_i\rangle\approx m(\bmod 1)/2$. This multi-key-switching algorithm inputs an extended ciphertext $\overline{ct}=(b,\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k)\in\mathcal{T}^{kn''+1}$ corresponding to $\bar{\boldsymbol{t}}=(\boldsymbol{t}_1,\ldots,\boldsymbol{t}_k)$ and a sequence of key-switching keys from $\boldsymbol{t}_i$ to $\boldsymbol{s}_i$, and returns an encryption of the same message under $\bar{\boldsymbol{s}}=(\boldsymbol{s}_1,\ldots,\boldsymbol{s}_k)$.

2.2.2 RGSW scheme

RGSW.Setup$(1^\lambda)\to(pp^{\mathrm{RGSW}})$: Input security parameter λ, output public parameter pp^{RGSW}.

RGSW.KeyGen$(pp^{\mathrm{RGSW}})\to(z,\boldsymbol{P})$: Input public parameter pp^{RGSW}, output public key $\boldsymbol{P}\in T^{d\times 2}$ and secret key z, where $T=\mathcal{T}[X]/\langle X^{N'}+1\rangle$, where N' is the degree of the polynomial.

RGSW.UniEnc$(\mu,z,\boldsymbol{P})\to(\boldsymbol{C},\boldsymbol{D},\boldsymbol{F})$: Input plaintext μ, secret key z, and public key $\boldsymbol{P}$, output uni-encryption ciphertext $(\boldsymbol{C},\boldsymbol{D},\boldsymbol{F})$.

$\overline{\boldsymbol{C}}\leftarrow$ RGSW.Expand$((\boldsymbol{C},\boldsymbol{D},\boldsymbol{F}),i,\{\boldsymbol{P}_j\}_{j\in[1,k]})$: Input ciphertext $(\boldsymbol{C},\boldsymbol{D},\boldsymbol{F})$, an index for user i, and a sequence of public keys $\{\boldsymbol{P}_j\}_{j\in[1,k]}$, output expanded ciphertext $\overline{\boldsymbol{C}}$.

RGSW.ExtProd$(\bar{\boldsymbol{c}},\overline{\boldsymbol{C}})\to\overline{\boldsymbol{c}'}$: Input expanded GSW ciphertext $\overline{\boldsymbol{C}}$ and expanded RLWE ciphertext $\bar{\boldsymbol{c}}$, output extension product ciphertext $\overline{\boldsymbol{c}'}$, where the RLWE ciphertext is a pair $(b(X),a(X))\in R$, $R=\mathbf{Z}[X]/\langle X^{N'}+1\rangle$, and expanded RLWE ciphertext with respect to the concatenated RLWE secret $\boldsymbol{z}=(1,z_1,\ldots,z_k)\in R^{k+1}$.

2.2.3 CCS19 scheme

Given the LWE dimension n' , key distribution χ, error parameter α, decomposition base B_{ks} and degree d_{ks}, secret parameter λ, RLWE dimension N', which has a power of two, the key distribution ψ over R, the error parameter β, the base integer $B\geqslant 2$, and the decomposition degree d, we generate a random vector $\boldsymbol{a}\leftarrow U(T^d)$.

(1) MKHE.Setup(1^λ):

– Run LWE.Setup(1^λ) to generate the parameter pp^{LWE}.

– Run RGSW.Setup(1^λ) to generate the parameter pp^{RGSW}.

– Return the generated public parameters $pp=(pp^{\mathrm{LWE}},pp^{\mathrm{RGSW}})$.

(2) MKHE.KeyGen(pp):

– Run $(z,\boldsymbol{P})\leftarrow$ RGSW.KeyGen(pp^{RGSW}) and set the public key as PK $=\boldsymbol{P}$. We write $\boldsymbol{t}=(z_0,-z_{N'-1},\ldots,-z_1)\in\mathbf{Z}^{N'}$ for $z(X)=z_0+z_1X+\cdots+z_{N'-1}X^{N'-1}$.

– Sample the LWE secret $\boldsymbol{s}=(s_1,\ldots,s_{n'})\leftarrow\psi$.

– Generate ciphertext $(\boldsymbol{C}_l,\boldsymbol{D}_l,\boldsymbol{F}_l)\leftarrow$ RGSW.UniEnc$(s_l,z,\boldsymbol{P})$ for $l\in[1,n']$ and set the bootstrapping key as $BK=(\boldsymbol{C}_l,\boldsymbol{D}_l,\boldsymbol{F}_l)$.

– Generate the key-switching key $KS\leftarrow$ LWE.KSGen$(\boldsymbol{t},\boldsymbol{s})$.

– Return the secret key $\boldsymbol{s}$. Publish the triple (PK,BK,KS) of the public, bootstrapping, and key-switching keys, respectively.

(3) MKHE.Enc(m): For an input bit $m\in\{0,1\}$, run LWE.Enc(m) and return an LWE encryption with the scaling factor 1/4. The output ciphertext $ct=(b,\boldsymbol{a})\in\mathcal{T}^{n'+1}$ satisfies $b+\langle\boldsymbol{a},\boldsymbol{s}\rangle\approx m(\bmod 1)/4$.

(4) MKHE.Dec($\overline{ct}, \{\boldsymbol{s}_j\}_{j\in[1,k]}$): For a ciphertext $\overline{ct} = (b, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k) \in \mathcal{T}^{kn'+1}$ and a tuple of secrets $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$, return the bit $m \in \{0,1\}$ that minimizes $|b + \sum_{j=1}^{k} \langle \boldsymbol{a}_j, \boldsymbol{s}_j \rangle - m/4|$.

(5) MKHE.NAND($\overline{ct_1}, \overline{ct_2}, \{(PK_j, BK_j, KS_j)\}_{j\in[1,k]}$): This function input two LWE ciphertexts $\overline{ct_1} \in \mathcal{T}^{k_1 n'+1}$ and $\overline{ct_2} \in \mathcal{T}^{k_2 n'+1}$, where k_i is the set of indices of the parties that are involved in either $\overline{ct_1}$ or $\overline{ct_2}$. In this function, PK_j, BK_j, and KS_j denote public key, bootstrapping key, and key-switching key, respectively.

This function consists of three steps. The first step expands the input LWE ciphertexts and evaluates the NAND gate homomorphically over the encrypted plaintexts.

Step 1.1: Extend $\overline{ct_1} \in \mathcal{T}^{k_1 n'+1}$ and $\overline{ct_2} \in \mathcal{T}^{k_2 n'+1}$ to the ciphertexts $\overline{ct_1}' \in \mathcal{T}^{kn'+1}$ and $\overline{ct_2}' \in \mathcal{T}^{kn'+1}$, respectively, by addtion 0 in corresponding position. The extended ciphertexts correspond to the concatenated secret key $\bar{\boldsymbol{s}} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k) \in \mathbf{Z}^{kn'}$, and the plaintexts remain unchanged. This is simply done by rearranging the components and putting zeros in the empty slots.

Step 1.2: Compute $\overline{ct}' = (5/8, 0, \ldots, 0) - \overline{ct}_1' - \overline{ct}_2' (\mathrm{mod}\ 1)$.

Step 2.1: For $i \in [1,k]$ and $l \in [1,n']$, run $\overline{\boldsymbol{C}_{i,l}} \leftarrow$ RGSW.Expand($(\boldsymbol{C}_{i,l}, \boldsymbol{D}_{i,l}, \boldsymbol{F}_{i,l}), \{\boldsymbol{P}_j\}_{j\in[1,k]}$) to generate an RGSW encryption of $s_{i,l}$ under the secret $\bar{\boldsymbol{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$. Return the shared bootstrapping key $\overline{BK} := \{\overline{\boldsymbol{C}_{i,l}}\}_{i\in[1,k], l\in[1,n']}$.

Step 2.2: Let $\overline{ct'} = (b', \boldsymbol{a}_1', \ldots, \boldsymbol{a}_k') \in \mathcal{T}^{kn'+1}$. Compute $\tilde{b} = 2N' \cdot b'$ and $\tilde{\boldsymbol{a}}_i = 2N' \cdot \boldsymbol{a}_i'$. Initialize the RLWE ciphertext as $ACC = (-h(X) \cdot X^{\tilde{b}}/8, \boldsymbol{0}) \in T^{k+1}$, where $h(X) = 1 + X + \cdots + X^{\frac{N'}{2}-1} - X^{\frac{N'}{2}+1} - \cdots - X^{N'-1}$.

Step 2.3: Let $\tilde{\boldsymbol{a}}_i = (\tilde{a}_{i,l})_{l\in[1,n]}$ for $i \in [1,k]$. Compute $ACC \leftarrow ACC + \text{RGSW.ExtProd}((X^{\tilde{a}_{i,l}} - 1) \cdot ACC, \overline{\boldsymbol{C}_{i,l}})$ recursively for all $i \in [1,k]$ and $l \in [1,n']$.

Step 2.4: Return $ACC \leftarrow (1/8, \boldsymbol{0}) + ACC (\mathrm{mod}\ 1)$.

Step 3.1: For $ACC = (c_0, c_1, \ldots, c_k) \in T^{k+1}$, let b'' be the constant term of c_0 and $\boldsymbol{a}_i''$ be the coefficient vector of c_i for $i \in [1,k]$. Construct the LWE ciphertext $\overline{ct''} = (b'', a_1'', \ldots, \boldsymbol{a}_k'') \in \mathcal{T}^{kn'+1}$.

Step 3.2: Let $\overline{KS} = \{KS_i\}_{i\in[1,k]}$. Run the multi-key-switching algorithm and return the ciphertext $\overline{ct} \leftarrow$ LWE.MKSwitch($\overline{ct}'', \overline{KS}$).

2.3 Functions for determining location of disease-causing genes

In this part, we introduce three location functions in JWB+17 that are used to locate the positions of disease-causing genes.

(1) MAX function

The MAX function studies target patients diagnosed with the same genetic disease and locates the most frequently mutated genes in all patients. This function sums up the genetic data of all patients (after preprocessing), with the position of the gene corresponding to the maximum summed value marked as 1 and the rest as 0. For the case of two participants, with inputs of n-bit genetic data $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$, the MAX function outputs $\boldsymbol{b} = (b_1, \ldots, b_n)$, where $b_i = \text{EQ}(\text{MAX}(\text{ADD}(x_1, y_1), \ldots, \text{ADD}(x_n, y_n)), \text{ADD}(x_i, y_i))$. ADD$(\cdot)$ outputs the n-bit sum of the two input genetic data, MAX$(\cdot)$ obtains the maximum value of the n input data, EQ$(\cdot)$ outputs 1 if the two input data are equal and 0 if not. Figure 2 shows an example of the MAX function.

For Boolean circuit implementation, the MAX function is realized by the addition circuit "ADD", max circuit "MAX", and equality circuit "EQ" (all of which can be constructed by the NAND gate). Figure 3 shows the specific circuit structure.

(2) Intersection function

The intersection function can be used to locate a commonly mutated gene in patients with the same genetic disease or who shows the same symptoms, with only the position of the mutated gene that appears in all the involved patients marked as 1. For the case of two participants, with inputs of n-bit genetic data $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$, the intersection function outputs $\boldsymbol{b} = (b_1, \ldots, b_n)$, where

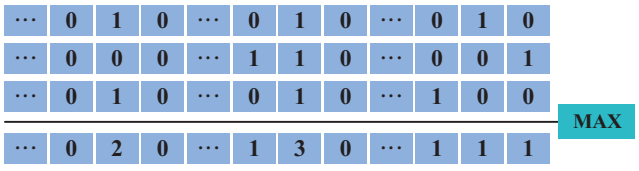

Fig. 2 Example of MAX function.

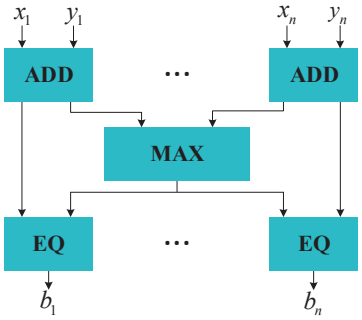

Fig. 3 MAX circuit.

$b_i = \text{AND}(x_i, y_i)$. Figure 4 shows an example of the intersection function.

For Boolean circuit implementation, the intersection function is realized by simple "AND" circuits, the specific structure of which is presented in Fig. 5.

(3) SET DIFF function

The SET DIFF function is used to locate a mutated gene that appears only in the child of a family, with the position of this gene marked as 1. For case of a family (father, mother, and a child), given the input of the parents' n-bit genetic data $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$, and child's genetic data $\boldsymbol{z} = (z_1, \ldots, z_n)$, the SET DIFF function outputs $\boldsymbol{b} = (b_1, \ldots, b_n)$, where b_i =AND(z_i, EQ(0, ADD(x_i, y_i))). Figure 6 shows an example of the SET DIFF function.

For Boolean circuit implementation, the SET DIFF function is realized by Boolean building blocks, including the addition circuit "ADD" and the equality circuit "EQ". Figure 7 shows the specific circuit structure.

3 Secure Scheme for Locating Disease-Causing Genes

3.1 Circuit optimization

(1) Optimization of SET DIFF circuit

Here, we show that the original circuit structure of the SET DIFF function, which requires complicated circuit

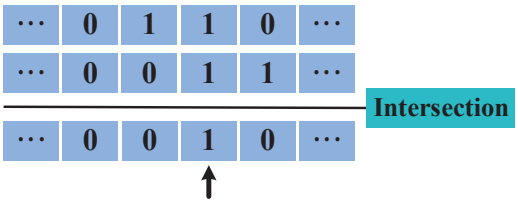

Fig. 4 Example of intersection function.

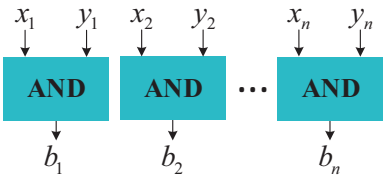

Fig. 5 Intersection circuit.

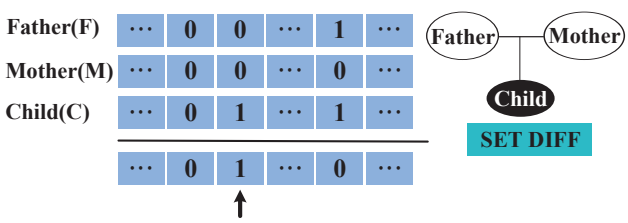

Fig. 6 Example of SET DIFF function.

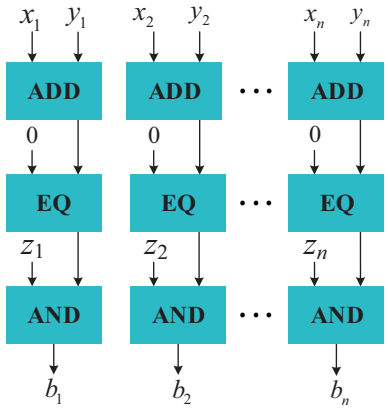

Fig. 7 SET DIFF circuit.

building blocks, i.e., the addition circuit "ADD" and equality circuit "Q", can be simplified by the use of brief "AND" and "NOT" circuits, thus greatly reducing the complexity of the circuit structure and improving the implementation efficiency of the SET DIFF function. The optimized circuit implementation of the SET DIFF function is as follows:

For the case of a family (father, mother, and a child), given the inputs of two parents' n-bit genetic data $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$, and the child's genetic data $\boldsymbol{z} = (z_1, \ldots, z_n)$, the improved SET DEFF function outputs $\boldsymbol{b} = (b_1, \ldots, b_n)$, where $b_i = \text{AND}(\text{AND}(-x_i, -y_i), z_i)$. Figure 8 shows the specific circuit structure.

(2) TH-intersection circuit

A malignant mutation at one or more sites in a gene can cause a genetic disease. For example, Parkinson's disease can be caused by one or more malignant mutations among 135 gene sites, but the three functions used in [JWB+17] can only be used to locate monogenic diseases. We present a TH-intersection function that can be used to locate polygenic diseases by setting an appropriate threshold, thereby extending the application scope of location functions.

In this circuit, given the inputs of m n-bit genetic data

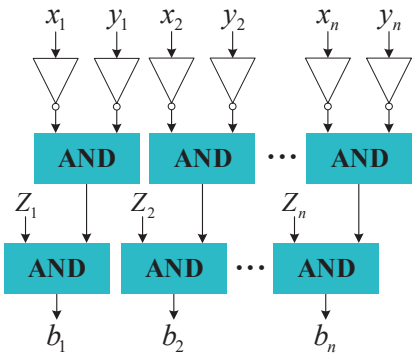

Fig. 8 Improved SET DIFF circuit.

$\boldsymbol{y}_j = (y_{j,1}, \ldots, y_{j,n})$, where $j \in [1, m]$, $\boldsymbol{z} = (z_1, \ldots, z_n)$ is output, where $z_i = \mathrm{LT}(l, \mathrm{ADD}(y_{1,i}, \ldots, y_{m,i}))$. The "LT" circuit implements less-than operations, which outputs 1 when the left-side input is less than the right-side inputs. Figure 9 shows the specific circuit structure of the TH-intersection function.

(3) Top-q circuit

In addition to the TH-intersection, we introduce another circuit called "Top-q", which outputs the Top-q gene positions with a high frequency of common mutation among all patients. As such, it can also be used to diagnose polygenic diseases.

In this circuit, h users' n-bit genetic vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,n})$, where $i \in [1, h]$, are input to obtain output $\boldsymbol{b} = (b_{\text{top-}q,1}, \ldots, b_{\text{top-}q,n})$. The specific steps of Top-q circuit are listed in Algorithm 1.

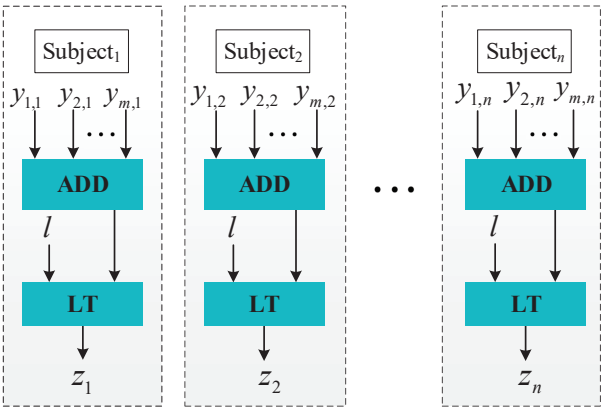

Fig. 9 TH-intersection circuit.

Algorithm 1 Top-q circuit

Input:

h user's n-bit genetic vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,n})$, where $i \in [1, h]$

Output:

$\boldsymbol{b} = (b_{\text{top-}q,1}, \ldots, b_{\text{top-}q,n})$

for each $j \in [1, n]$ **do**
  $\bar{x}_{\text{top-}1,j} = \mathrm{ADD}(\boldsymbol{x}_{1,j}, \ldots, \boldsymbol{x}_{h,j})$;
end for
for each $r \in [1, q-1]$ **do**
  for each $j \in [1, n]$ **do**
    $\bar{b}_{\text{top-}r,j} = \mathrm{EQ}(\mathrm{MAX}(\bar{x}_{\text{top-}r,1}, \ldots, \bar{x}_{\text{top-}r,n}), \bar{x}_{\text{top-}r,j})$;
    $\bar{x}_{\text{top-}(r+1),j} = \prod_{j=1}^{r} (1 - \bar{b}_{\text{top-}j,j}) \bar{x}_{\text{top-}r,j}$;
  end for
end for
for each $j \in [1, n]$ **do**
  $\bar{b}_{\text{top-}q,j} = \mathrm{EQ}(\mathrm{MAX}(\bar{x}_{\text{top-}q,1}, \ldots, \bar{x}_{\text{top-}q,n}), \bar{x}_{\text{top-}q,j})$;
  $b_{\text{top-}q,j} = 1 - \prod_{l=1}^{k} (1 - \bar{b}_{\text{top-}l,j})$;
end for
return $\boldsymbol{b} = (b_{\text{top-}q,1}, \ldots, b_{\text{top-}q,n})$;

Figure 10 shows the specific structure of the Top-q circuit.

3.2 Direct decryption protocol for CCS19

MKHE can be applied to realize secure computation among multiple parties, and the evaluated ciphertext can be jointly decrypted by all involved users. However, sometimes it is not preferred that the final decryption result be known by all involved users, but rather just designated user recognized as legitimate, perhaps even user who had not participated in the computing process. For this scenario, as shown in the schematic in Fig. 11, a directed decryption protocol is necessary to enable the data owners to control their plaintext[15].

We constructed a directed decryption protocol by adding the ciphertext of 0 (under the public key of the target user) to the intermediate decryption result of the users involved in the homomorphic evaluation. The process of the directed decryption protocol MKHE.DirDec( ) is as

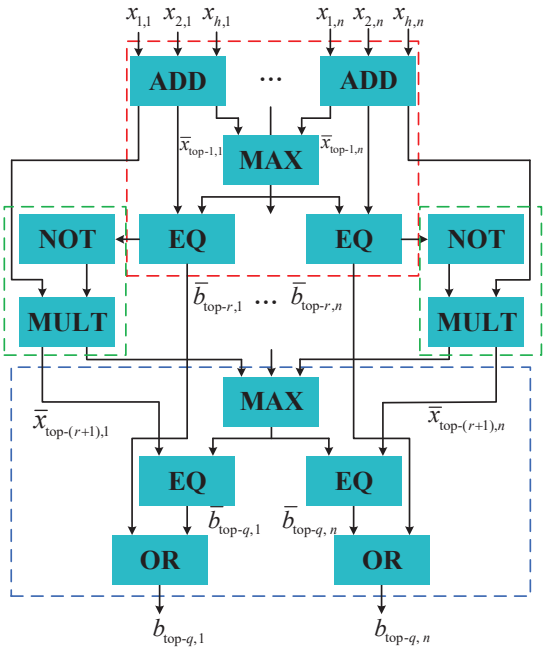

Fig. 10 Top-q circuit.

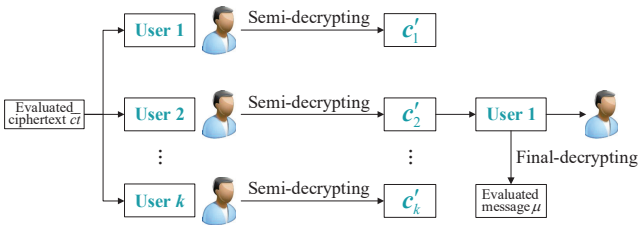

Fig. 11 Schematic of the directed decryption process in our MKHE scheme.

follows:

MKHE.SemiDec $(\bar{\boldsymbol{c}}, \boldsymbol{s}_i)$: For an extended ciphertext $\bar{\boldsymbol{c}} = (b, \boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k) \in \mathcal{T}^{kn'+1}$ and the i-th secret $\boldsymbol{s}_i$, where $1 \leqslant i \leqslant k$, user i does as follows:

(1) Generate target user's ciphertext of 0: Input the target public key of user i^*, and compute

$$\boldsymbol{c}_{i^*} = \text{MKHE.Enc}(pk_{i^*}, 0) = (b_{i^*}, \boldsymbol{a}_{i^*}) \in \mathcal{T}^{n'+1},$$

note that $b_{i^*} = \boldsymbol{a}_{i^*}\boldsymbol{s}_{i^*} + e$, where e is a small error.

(2) Output $\boldsymbol{c}'_i = (b'_i, \boldsymbol{a}'_i) := (b_{i^*} + \langle \boldsymbol{a}_i, \boldsymbol{s}_i \rangle, \boldsymbol{a}_{i^*}) \in \mathcal{T}^{n'+1}$.

MKHE.FinalDec$(b, \{\boldsymbol{c'}_i\}_{i\in[1,k]})$: For the first entry b of an input extended ciphertext $\bar{\boldsymbol{c}} = (b, \boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k) \in \mathcal{T}^{kn'+1}$ and the partial decryption ciphertexts$\{\boldsymbol{c'}_i\}_{i\in[1,k]}$, output the bit $m \in \{0, 1\}$ which minimizes $|(b + \sum_{i=1}^{k} b'_i) - \langle \sum_{i=1}^{k} \boldsymbol{a}'_i, \boldsymbol{s}_{i^*} \rangle - m/4|$.

Lemma 1: Let B denote the bound of noise e_i in a fresh LWE ciphertext $\boldsymbol{c}'_i$ and $\bar{B}$ denote the bound of noise $\bar{e}$ in the input ciphertext $\bar{\boldsymbol{c}}$, then the directed decryption process is correct if

$$|\bar{e} + e_1 + \cdots + e_k| \leqslant \bar{B} + kB < q/4.$$

Note that in most current MKHE schemes (including the original CCS19), the results of homomorphic evaluations can only be finally decrypted by users who were involved in the evaluation process. The directed decryption protocol designed in this paper allows for the resulting ciphertext to be decrypted by any legitimate users. Moreover, as no homomorphic multiplication is involved in our protocol, there is no need for the use of techniques to control associated noise.

3.3 Secure scheme for locating disease-causing genes based on optimized CCS19

In this section, we introduce our secure scheme for locating disease-causing genes, the main steps of which are as follows:

(1) Set up: Generate the basic parameters of the MKHE scheme (CCS19): Medical institutions i first generate their keys (including public key pk_i, secret key sk_i, and evaluation key evk_i), where $i \in [1, N]$, and N is the number of medical institutions involved in the scheme, to obtain the gene mutation database.

Preprocessing of genetic data: The medical institution i creates a mutated bit vector for their research subject, and compares the gene data of this research subject with those in the genetic mutation database. A bit-string $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n}) \in \mathbf{Z}_2^n$ is obtained that contains genetic variation information for the research subject, where $i \in [1, N]$, and n is the number of items in the genetic mutation database (for the convenience of description, it is assumed that each medical institution has only one research subject participating in the calculation).

(2) Encryption: The medical institution i encrypts the genetic data $\boldsymbol{x}_i$ by pk_i, and obtains $\{c_{i,j} \leftarrow \text{MKHE.Enc}(pk_i, x_{i,j})\}_{j\in[1,n]}$, then uploads the ciphertext to the cloud, where $i \in [1, N]$.

(3) Homomorphic evaluations: After the cloud receives the encrypted genetic data $\{c_{i,j}\}_{i\in[1,N],j\in[1,n]}$, it first extends them to $\{\bar{c}_{i,j} \leftarrow \text{MKHE.Extend}(c_{i,j}, (pk_1, \ldots, pk_N), em_i)\}_{i\in[1,N],j\in[1,n]}$, which corresponds to the secret key $sk := (sk_1 | sk_2 | \cdots | sk_N)$, and implements homomorphic evaluations on $\{\bar{c}_{i,j}\}_{i\in[N],j\in[n]}$ using the location function. It then obtains evaluation result $\{c'_j \leftarrow \text{MKHE.Eval}(\mathcal{C}, (\bar{c}_{1,j}, pk_1, evk_1), \ldots, (\bar{c}_{N,j}, pk_N, evk_N))\}_{j\in[1,n]}$, and transfers it to all the involved medical institutions.

(4) Decryption: When all the involved medical institutions receive $\{c'_j\}_{j\in[1,n]}$, they implement the direct decryption protocol described in Section 3.2 to obtain the desired information about the mutation position that may be causing a genopathy: $\{x_j \leftarrow \text{MKHE.DirDec}((sk_1, sk_2, \ldots, sk_N), c'_j)\}_{j\in[1,n]}$. Figure 12 shows the secure locating pathogenic genes.

The main difference between our scheme and CCS19 is the decryption method, which addresses the security issues associated with the direct decryption protocol, as reported in Ref. [15]. Our scheme is semantically secure under (R)LWE assumptions, and also requires a circular security assumption that is identical to that of CCS19.

4 Experimental Result

We implemented the secure protocol for locating pathogenic genes for the intersection and SET DIFF circuits involving two parties, and the TH-intersection circuit involving three parties, for which 48-bit genetic data were input for each party. Table 1 presents the results. The experimental environment included a notebook computer of Dell Precision 7530 and a Ubuntu 18.04 Stl computer system, CPU: Intel(R) Core(TM) i7-8750h, memory: 16 GB.

We used the same LWE and RLWE parameters as those used in the Tours FHE (TFHE) and CCS19 implementations, which achieved a 152-bit security level, based on the security analysis reported in Ref. [19]. The dimension of LWE $n = 500$, the noise parameter of LWE $\alpha = 2.43 \times 10^{-5}$, the degree of polynomial in RLWE $N = 1024$, and the noise parameter of RLWE $\beta = 3.29 \times 10^{-10}$. The term pre-time indicates

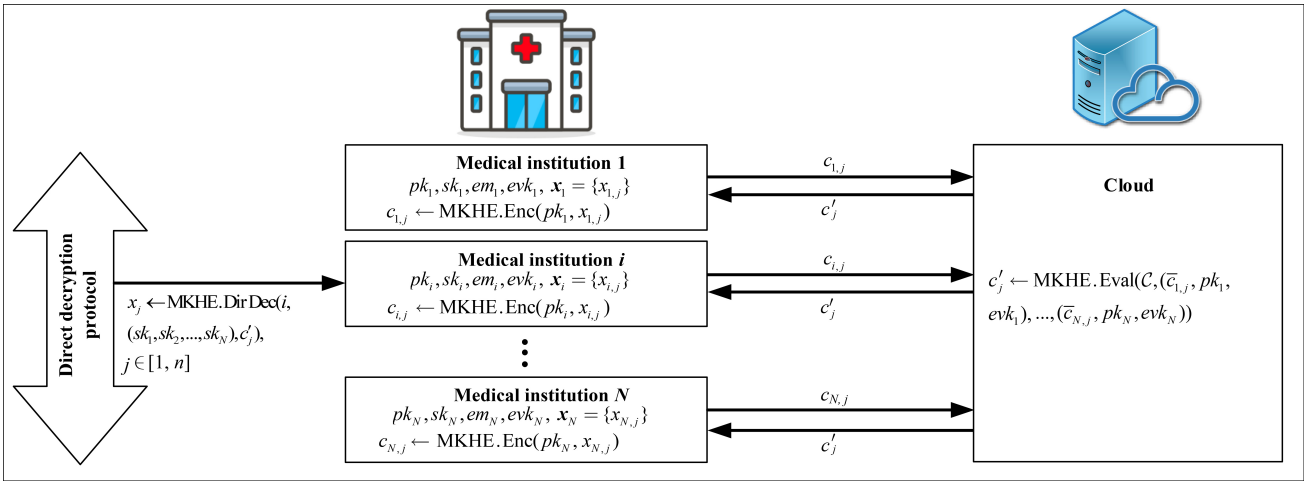

Fig. 12 Secure protocol for locating pathogenic genes.

Table 1 Comparison of communication traffic and running time of JWB + 17 and our scheme.

Scheme	Preprocessing running time (s)	Preprocessing communication traffic (KB)	Intersection		SET DIFF		TH-intersection	
			Communication traffic (KB)	Running time (s)	Communication traffic (KB)	Running time (s)	Communication traffic (KB)	Running time (s)
JWB+17	0	0	144.22	0	1360.80	0	Not available	Not available
Our scheme	2.03 (two parties) 3.10 (three parties)	2	7.90	0.05	7.90	0.98	17.90	1.03

the preprocessing running time, and pre-communication indicates the preprocessing communication traffic.

The experimental results show that our MKFHE-based scheme has the advantage of low communication traffic, and the disadvantage of being somewhat slow. Our communication traffic is reduced by one or two orders of magnitude compared to JWB+17, but the slow running time of our scheme is due to its complicated homomorphic evaluation process. As such, our scheme is suitable for systems with limited bandwidth. The JWB+17 tries to reduce its bandwidth using the "two-cloud" model, which requires a strong assumption that the two-cloud servers are non-colluding, which is impractical in the real world. In addition, the gabled circuits of Yao's protocol in JWB+17 cannot be reused, so the system must construct a new gabled circuit when starting a new computation. In our scheme, the ciphertext of the genetic data from all involved patients can be reused throughout the process, and each medical institution involved must only upload their encrypted genetic data once. Thus, our scheme supports more convenient offline operations. Table 2 shows an overall comparison of JWB+17 and our proposed scheme.

5 Conclusion

In this paper, we propose a secure frequency-based scheme for locating disease-causing genes based on multi-key homomorphic encryption. To locate genes that may cause polygenic diseases, our scheme homomorphically analyzes the encrypted genetic data of multiple medical institutions. Target users can also be chosen to receive the final result, thereby enhancing the ability of the data owner to control their genetic data. The communication cost of our solution is low, and this scheme is most suitable for systems with limited bandwidth.

Acknowledgment

This work was supported by the National Key

Table 2 Overall comparison of JWB + 17 and our scheme.

Scheme	Collusion attack	Ciphertext reuse	Offline operation	Communication traffic	Running time
JWB+17	×	×	×	×	✓
Our scheme	✓	✓	✓	✓	×

R&D Program of China (No. 2017YFB0802000), the Innovative Research Team in Engineering University of PAP (No. KYTD201805), the National Natural Science Foundation of China (No. 61872384), the Natural Science Basic Research Plan in Shaanxi Province of China (No. 2020JQ-492), and the Fundamental Research Project of Engineering University of PAP (Nos. WJY201910, WJY201914, and WJY201912).

References

[1] H. X. Tang, X. Q. Jiang, X. F. Wang, S. Wang, H. Sofia, D. Fox, K. Lauter, B. Malin, A. Telenti, L. Xiong, et al., Protecting genomic data analytics in the cloud: State of the art and opportunities, *BMC Med. Genomics*, vol. 9, no. 1, p. 63, 2016.

[2] J. W. Bos, K. Lauter, and M. Naehrig, Private predictive analysis on encrypted medical data, *J. Biomed. Inform.*, vol. 50, pp. 234–243, 2014.

[3] M. Kim, Y. Song, and J. H. Cheon, Secure searching of biomarkers through hybrid homomorphic encryption scheme, *BMC Med. Genomics*, vol. 10, no. 2, p. 42, 2017.

[4] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano, Deriving genomic diagnoses without revealing patient genomes, *Science*, vol. 357, no. 6352, pp. 692–695, 2017.

[5] K. Lauter, A. López-Alt, and M. Naehrig, Private computation on encrypted genomic data, in *Proc. of Int. Conf. Cryptology and Information Security in Latin America*, Florianópolis, Brazil, 2014, pp. 3–27.

[6] M. Kim and K. Lauter, Private genome analysis through homomorphic encryption, *BMC Med. Inform. Decis. Mak.*, vol. 15, no. 5, p. S3, 2015.

[7] A. C. C. Yao, How to generate and exchange secrets, in *Proc. of 27th Annu. Symp. Foundations of Computer Science*, Toronto, Canada, 1986, pp. 162–167.

[8] A. López-Alt, E. Tromer, and V. Vaikuntanathan, On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption, in *Proc. 44th Annu. ACM Symp. Theory of Computing*, New York, NY, USA, 2012, pp. 1219–1234.

[9] P. Mukherjee and D. Wichs, Two round multiparty computation via multi-key FHE, in *Proc. of Annu. Int. Conf. Theory and Applications of Cryptographic Techniques*, Vienna, Austria, 2016, pp. 735–763.

[10] L. Chen, Z. F. Zhang, and X. Q. Wang, Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension, in *Proc. of Theory of Cryptography Conf.*, Baltimore, MD, USA, 2017, pp. 597–627.

[11] R. L. Rivest, L. Adleman, and M. L. Dertouzos, On data banks and privacy homomorphisms, in *Foundations of Secure Computation*. Orlando, FL, USA: Academia Press, 1978, pp. 169–180.

[12] C. Gentry, Fully homomorphic encryption using ideal lattices, in *Proc. of 41st Annu. ACM Symp. Theory of Computing*, Bethesda, MD, USA, 2009, pp. 169–178.

[13] G. L. Huang, T. F. Zhang, J. Cheng, Y. X. Zhou, C. X. Liu, G. F. Jin, M. X. Wu, Y. B. Yan, and R. Yang, Gene disease diagnostic system, *Tsinghua Science and Technology*, vol. 7, no. 6, pp. 665–667, 2002.

[14] J. W. Bos, K. Lauter, and M. Naehrig, Private predictive analysis on encrypted medical data, *J. Biomed. Inform.*, vol. 50, pp. 234–243, 2014.

[15] N. B. Li, T.P. Zhou, X. Y. Yang, Y. L. Han, W. C. Liu, and G. S. Tu, Efficient multi-key fhe with short extended ciphertexts and directed decryption protocol, *IEEE Access*, vol. 7, pp. 56724–56732, 2019.

[16] S. Wang, Y. C. Zhang, W. R. Dai, K. Lauter, M. Kim, Y. Z. Tang, H. K. Xiong, and X. Q. Jiang, HEALER: Homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS, *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2016.

[17] H. Chen, I. Chillotti, and Y. Song, Multi-key homomorphic encryption from TFHE, in *Proc. of Int. Conf. Theory and Application of Cryptology and Information Security*, Kobe, Japan, 2019, pp. 446–472.

[18] R. Lindner and C. Peikert, Better key sizes (and attacks) for LWE-based encryption, in *Proc. of Cryptographers' Track at the RSA Conf.*, San Francisco, CA, USA, 2011, pp. 319–339.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds, in *Proc. of Int. Conf. Theory and Application of Cryptology and Information Security*, Hanoi, Vietnam, 2016, pp. 3–33.

Tanping Zhou received the PhD degree from Engineering University of People's Armed Police in 2018. He is now an associate professor at Engineering University of People's Armed Police, and also a postdoctoral researcher at Institute of Software, Chinese Academy of Sciences. His main research interests include fully homomorphic encryption and encryption scheme based on lattice.

Wenchao Liu received MEng degree from Engineering University of People's Armed Police in 2020. He is now a lecturer at Engineering University of People's Armed Police. His main research interests include fully homomorphic encryption and information security.

Ningbo Li received the PhD degree from Engineering University of People's Armed Police in 2020. He is now a lecturer at People's Armed Police Command College China. His main research interests include fully homomorphic encryption and encryption scheme based on lattice.

Xiaoyuan Yang received the MEng degree from Xi'an Electronic Science and Technology University, Xi'an, China in 1991. He is now a professor and PhD supervisor at Engineering University of People's Armed Police. His main research interests include information security and cryptology.

Yiliang Han received the PhD degree in computer science from Xi'an Jiaotong University in 2020. He is now a professor and PhD supervisor at the Engineering University of People's Armed Police. His main research interests include information security and cryptology.

Shangwen Zheng received the BEng degree from Engineering University of People's Armed Police in 2019. He is now a master student at Engineering University of People's Armed Police. His main research interests include fully homomorphic encryption and information security.