The Author(s) *BMC Medical Genomics* 2017, **10**(Suppl 2):42
DOI 10.1186/s12920-017-0280-3

RESEARCH Open Access

Secure searching of biomarkers through hybrid homomorphic encryption scheme

Miran Kim[1*], Yongsoo Song[2] and Jung Hee Cheon[2]

From iDASH Privacy and Security Workshop 2016
Chicago, IL, USA. 11/11/2016

Abstract

Background: As genome sequencing technology develops rapidly, there has lately been an increasing need to keep genomic data secure even when stored in the cloud and still used for research. We are interested in designing a protocol for the secure outsourcing matching problem on encrypted data.

Method: We propose an efficient method to securely search a matching position with the query data and extract some information at the position. After decryption, only a small amount of comparisons with the query information should be performed in plaintext state. We apply this method to find a set of biomarkers in encrypted genomes. The important feature of our method is to encode a genomic database as a single element of polynomial ring.

Result: Since our method requires a single homomorphic multiplication of hybrid scheme for query computation, it has the advantage over the previous methods in parameter size, computation complexity, and communication cost. In particular, the extraction procedure not only prevents leakage of database information that has not been queried by user but also reduces the communication cost by half. We evaluate the performance of our method and verify that the computation on large-scale personal data can be securely and practically outsourced to a cloud environment during data analysis. It takes about 3.9 s to *search-and-extract* the reference and alternate sequences at the queried position in a database of size 4M.

Conclusion: Our solution for finding a set of biomarkers in DNA sequences shows the progress of cryptographic techniques in terms of their capability can support real-world genome data analysis in a cloud environment.

Keywords: Homomorphic encryption, Biomarkers

Background

The rapid development of genome sequencing technology enables us to access large genome dataset and it looks poised to make a significant breakthrough in medical research. While genomic data can be used for a wide range of applications including healthcare, biomedical research, and direct-to-consumer services, it has numerous special distinguishing features and it can violate personal privacy via genetic disclosure or genetic discrimination [1–3]. Due to these potential privacy issues, it should be managed with care.

There have been various privacy-enhancing techniques using cryptographic methods as outsourced analysis tools of genomic data. Recently, it has been suggested that we can preserve privacy through homomorphic encryption (HE), which allows computations to be carried out on ciphertexts. Yasuda et al. [4] gave a practical solution to find the location of a pattern in a text by computing multiple Hamming distance values on encrypted data. Lauter et al. [5] gave a solution to privately compute the basic genomic algorithms used in genome-wide association studies.

Homomorphic encryption can be applied to privacy-preserving sequence comparison, but it is still impractical for the analysis of entire human genome information. For example, Cheon et al. [6] presented a protocol to compute

*Correspondence: mrkim@ucsd.edu
[1]Division of Biomedical Informatics, University of California- San Diego, San Diego, CA 92093, USA
Full list of author information is available at the end of the article

the edit distance on homomorphically encrypted data but it took about 27 s even on length 8 DNA sequence. It is not easy to efficiently approximate the edit distance over encryption even though the distance to a public human DNA sequence is given [7]. This inefficiency comes from the difficulty of homomorphic evaluation of equality test: Encrypting the inputs bit-wise and computing over the encrypted bits yield expensive computation cost (at least linear in the data bit-length).

In this paper, we suggest an efficient method to securely search a set of biomarkers using hybrid Ring-GSW homomorphic encryption scheme.

Problem setting

The iDASH (Integrating Data for Analysis, 'anonymization' and SHaring) National Center organizes the iDASH Privacy & Security challenge for secure genome analysis. This paper is based on a submission to the task 3 in 2016 iDASH challenge: secure outsourcing of testing for genetic diseases on encrypted genomes. The goal of this task is to privately calculate the probability of genetic diseases through matching a set of biomarkers to encrypted genomes stored in a public cloud service. The requirement is that the entire matching process needs to be carried out using homomorphic encryption so that any information about database and query should not be revealed to the server during computation.

Suppose that the client has a Variation Call Format (VCF) file which contains genotype information such as chromosome number and position in the genome. It also contains some information for each position such as reference and alternate sequences, where each base must be one of SNPs: A, T, G, and C. The client encrypts the information using homomorphic encryption and the server calculates the exact match over the encrypted data. The outcome is the absence/presence of the specified biomarkers, that is, an encryption of 1 if matched; otherwise, an encryption of 0. Finally the client decrypts the result by the secret key of homomorphic encryption.

Practical homomorphic encryption

Fully Homomorphic cryptosystems allow us to homomorphically evaluate any arithmetic circuit without decryption. However, the noise of the resulting ciphertext grows during homomorphic evaluations, slightly with addition but substantially with multiplication. For efficiency reasons, for tasks which are known in advance, we use a more practical *Somewhat Homomorphic Encryption* (SHE) scheme, which evaluates functions up to a certain complexity. In particular, two techniques are used for noise management of SHE: one is the *modulus-switching* technique introduced by Brakerski, Gentry and Vaikuntanathan [8], which scales down a ciphertext during every multiplication operation and reduces the noise by its scaling factor. The other is a *scale-invariant* technique proposed by Brakerski such that the same modulus is used throughout the evaluation process [9].

Let us denote by $[\cdot]_Q$ the reduction modulo Q into the interval $(-Q/2, Q/2] \cap \mathbb{Z}$ of the integer or integer polynomial (coefficient-wise). For a security parameter λ, we choose an integer $M = M(\lambda)$ that defines the M-th cyclotomic polynomial $\Phi_M(X)$. For a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(\Phi_M(X))$, 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 R. We use the standard notation $a \leftarrow \mathcal{D}$ to denote that a is chosen from the distribution $\mathcal{D}$.

The basic scheme

The following is a description of basic homomorphic encryption scheme based on the hardness of (decisional) *Ring Learning with Errors* (RLWE) assumption, which was first introduced by Lyubashevsky et al. [10]. The assumption is that it is infeasible to distinguish the following two distributions. The first distribution consists of pairs (a_i, u_i), where a_i and u_i are drawn uniformly at random from $\mathcal{R}_Q$. The second distribution consists of pairs of the form (a_i, b_i)=$(a_i, a_i\mathsf{s} + e_i)$ where a_i is uniformly random in $\mathcal{R}_Q$ and s, e_i are drawn from the error distribution χ. To improve efficiency for HE, we use sparse secret keys s with coefficients sampled from $\{0, \pm 1\}$ as in [11].

- RLWE.ParamsGen(λ): Given the security parameter λ, choose an integer M, a modulus Q, a plaintext modulus t with $t|Q$, and discrete Gaussian distribution χ_{err}. Output $params \leftarrow (M, Q, t, \chi_{err})$.
- RLWE.KeyGen($params$): On the input parameters, let $N = \phi(M)$ and choose a sparse random s from $\{0, \pm 1\}^N$. Generate an RLWE instance $(a, b) = (a, [-a\mathsf{s} + e]_Q)$ for $e \leftarrow \chi_{err}$. We set the secret key $\mathsf{sk} \leftarrow \mathsf{s}$ and the public key $\mathsf{pk} \leftarrow (a, b)$.
- RLWE.Enc(m, pk): To encrypt $m \in \mathcal{R}_t$, choose a small polynomial v and two Gaussian polynomials e_0, e_1 over $\mathcal{R}$ and output the ciphertext
$$\begin{aligned}\mathsf{ct} &\leftarrow (c_0, c_1)\\ &= ((Q/t)m, 0) + (bv + e_0, av + e_1) \in \mathcal{R}_Q^2.\end{aligned}$$
- RLWE.Dec(ct, sk): Given a ciphertext $\mathsf{ct} = (c_0, c_1)$, output $m \leftarrow \lfloor (t/Q) \cdot [c_0 + \mathsf{s} \cdot c_1]_Q \rceil$.
- RLWE.Add(ct, ct′): Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{ct}' = (c_0', c_1')$, the homomorphic addition is computed by $\mathsf{ct}_{\mathsf{add}} \leftarrow \left([c_0 + c_0']_Q, [c_1 + c_1']_Q \right)$.

Throughout this paper, we assume that the integer M is a power of two so that $N = M/2$ and $\phi_M(X) = X^N + 1$. We adapt the conversion and modulus-switching techniques of [12]. The conversion algorithm changes an RLWE encryption of $m = \sum_i m_i X^i$ into an LWE encryption of its constant term m_0, and the modulus switching reduces the ciphertext modulus Q down to q while preserving the message. We note that an LWE ciphertext is represented as a vector in $\mathbb{Z}_q$ for some modulus q, and the decryption procedure is done by an inner product of the ciphertext and the secret key vector.

- RLWE.Conv(ct): Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ with $c_0 = \sum_i c_{0,i}X^i$ and $c_1 = \sum_i c_{1,i}X^i$, output the vector $\mathsf{ct}' = (c_{0,0}, c_{1,0}, -c_{1,N-1}, \ldots, -c_{1,1})$.
- LWE.ModSwitch(ct): Given a ciphertext $\mathsf{ct} \in \mathbb{Z}_Q^{N+1}$, output the vector $\mathsf{ct}' \leftarrow \lfloor (q/Q) \cdot \mathsf{ct} \rceil \in \mathbb{Z}_q^{N+1}$.

An RLWE ciphertext $\mathsf{ct} = (c_0, c_1)$ has the decryption structure of the form $c_0 + c_1 \cdot \mathsf{s} = (Q/t) \cdot m + e$ and its constant term is

$$c_{0,0} + c_{1,0}s_0 - \sum_{i=1}^{N-1} c_{1,N-i}s_i = (Q/t) \cdot m_0 + e_0.$$

It can be represented as an inner product of a vector $(c_{0,0}, c_{1,0}, -c_{1,N-1}, -c_{1,N-2}, \ldots, -c_{1,1})$ and the desired LWE secret key $\vec{s} = (1, s_0, \ldots, s_{N-1})$. Hence the output of the conversion algorithm can be seen as an LWE encryption of m_0. It is also easy to check that if $\mathsf{ct} \in \mathbb{Z}_Q^{N+1}$ satisfies $\langle \mathsf{ct}, \vec{s} \rangle = (Q/t) \cdot m + e \pmod{Q}$, then the output of LWE.ModSwitch algorithm satisfies $\langle \mathsf{ct}', \vec{s} \rangle = (q/t) \cdot m + e' \pmod{q}$ for some $e' \approx (q/Q) \cdot e$. These techniques have been proposed for an efficient bootstrapping [12], but they will play totally different roles in our application. Finally an LWE ciphertext of modulus q can be decrypted by $\vec{s}$ as follows.

- LWE.Dec(ct, sk): Given a ciphertext $\mathsf{ct} \in \mathbb{Z}_q^{N+1}$, output the value $m \leftarrow \lfloor (t/q) \cdot [\langle \mathsf{ct}, \vec{s} \rangle]_q \rceil$.

If $\langle \mathsf{ct}, \vec{s} \rangle = (q/t) \cdot m + e \pmod{q}$ for some small enough e, it returns the correct message m modulo t. More precisely, the decryption procedure works if $|te/q| < 1/2$.

The Ring-GSW scheme

Gentry et al. [13] suggested a fully homomorphic encryption based on the LWE problem, where the message is encrypted as an approximate eigenvalue of a ciphertext. Ducas and Micciancio [12] described its RLWE variant. The RGSW symmetric encryption scheme consists of the following algorithms.

- RGSW.ParamsGen(·), RGSW.KeyGen(·): Use the same parameter *params* and secret key s with the basic RLWE scheme. Additionally set the decomposition base B_{g} and exponent d_{g} satisfying $B_{\mathsf{g}}^{d_{\mathsf{g}}} \geq Q$.
- RGSW.Enc(m, sk): To encrypt $m \in \mathcal{R}_t$, pick a matrix $\mathbf{a} \in \mathcal{R}_Q^{2d_{\mathsf{g}}}$ uniformly at random, and $\mathbf{e} \in \mathcal{R}^{2d_{\mathsf{g}}} \simeq \mathbb{Z}^{2d_{\mathsf{g}} \cdot n}$ with discrete Gaussian distribution χ of parameter ς, and output the ciphertext

$$\mathsf{CT} \leftarrow [\mathbf{b}, \mathbf{a}] + m\mathbf{G} \in \mathcal{R}_Q^{2d_{\mathsf{g}} \times 2}$$

where $\mathbf{b} = -\mathbf{a} \cdot \mathsf{s} + \mathbf{e}$ and the gadget matrix $\mathbf{G} = \left(\mathbf{I} \parallel B_{\mathsf{g}}\mathbf{I} \parallel \ldots \parallel B_{\mathsf{g}}^{d_{\mathsf{g}}-1}\mathbf{I}\right)^T \in \mathcal{R}_Q^{2d_{\mathsf{g}} \times 2}$ for 2×2 identity matrix $\mathbf{I}$.

Let $\mathsf{WD}_{B_{\mathsf{g}}}(\cdot)$ be the decomposition with the base B_{g}, where the dimension of input vector is multiplied by d_{g} through this algorithm. The RGSW encryption of m satisfies $\mathsf{CT} \cdot (1, \mathsf{s}) = m \cdot \left(1, \mathsf{s}, \ldots, B_{\mathsf{g}}^{d_{\mathsf{g}}-1}, B_{\mathsf{g}}^{d_{\mathsf{g}}-1}\mathsf{s}\right) + \mathsf{e}$. Roughly, m is an *approximate* eigenvalue of $\mathsf{WD}_{B_{\mathsf{g}}}(\mathsf{CT})$ with respect to the eigenvector $\left(1, \mathsf{s}, \ldots, B_{\mathsf{g}}^{d_{\mathsf{g}}-1}, B_{\mathsf{g}}^{d_{\mathsf{g}}-1}\mathsf{s}\right)$. In [14], the hybrid multiplication between RGSW ciphertexts and RLWE ciphertexts has been defined as follows.

- Hybrid.Mult(CT, ct): Given an RGSW ciphertext $\mathsf{CT} \in \mathcal{R}_Q^{2d_{\mathsf{g}} \times 2}$ and an RLWE ciphertext $\mathsf{ct} \in \mathcal{R}_Q^2$ output the vector $\mathsf{ct}' \leftarrow \mathsf{CT}^T \cdot \mathsf{WD}_{B_{\mathsf{g}}}(\mathsf{ct})$.

If CT and ct are RGSW and RLWE encryptions of m and m', respectively, their multiplication ct' is a valid RLWE encryption of mm'. For convenience, we will denote Hybrid.Mult(CT, ct) algorithm by $\boxdot$, *i.e.*, $(\mathsf{CT}, \mathsf{ct}) \in \mathcal{R}_Q^{2d_{\mathsf{g}} \times 2} \times \mathcal{R}_Q^2 \mapsto \mathsf{CT} \boxdot \mathsf{ct} \in \mathcal{R}_Q^2$.

Methods

Privacy-preserving database searching and extraction

Let us consider a database of a set of n tuples. Each tuple consists of pairs (d_i, α_i) for $i = 1, \ldots, n$, where d_i denotes a *data-tag* in the domain $\{0, 1, \ldots, \mathcal{T} - 1\}$ and α_i represents the corresponding *value attribute* in a plaintext space $\mathbb{Z}_t \backslash \{0\}$.Note that all the tags should be distinct from each other. For instance, in the case of personal information database, α_i may be the age of user whose identity number is d_i.

Given a *query tag* d from a tag domain and a *query value* α from a plaintext space, the matching problem is to determine the existence of an index i such that $(d, \alpha) = (d_i, \alpha_i)$. Now consider the following simplified search query: select α_i if there exists an index i such that $d_i = d$; otherwise zero ($\perp$). The purpose of this section is to store the database and carry out this search query on the public cloud. The server should learn nothing from encrypted query and any information other than the final result should not be leaked to user. Throughout this work, we will use semi-honest (honest but curious) adversary model, which is a standard assumption for evaluation of homomorphic encryption.

Our main idea is the following encoding method of database suitable for the efficient computation of equality test and extraction:

$$\mathsf{DB}(X) = \sum_i \alpha_i X^{d_i} \in \mathbb{Z}_t[X].$$

The user encrypts this polynomial with the RLWE public-key encryption scheme and stores the ciphertext $\mathsf{ct}_{\mathsf{DB}}$ in the server. At the query phase, given a query tag d, the user encrypts the monomial X^{-d} with the RGSW symmetric encryption scheme and sends the ciphertext CT_{Q} to the server. We assume that the RGSW encryption scheme has the same secret key sk as the one of RLWE encryption scheme.

Given two ciphertexts $\mathsf{CT}_{\mathsf{Q}} \leftarrow \mathsf{RGSW.Enc}\left(X^{-d}\right)$ and $\mathsf{ct}_{\mathsf{DB}} \leftarrow \mathsf{RLWE.Enc}(\mathsf{DB}(X))$, the server first performs their multiplication to obtain an ciphertext, denoted by $\mathsf{ct}_{\mathsf{mult}} = \mathsf{CT}_{\mathsf{Q}} \boxdot \mathsf{ct}_{\mathsf{DB}}$. It follows from the previous section that $\mathsf{ct}_{\mathsf{mult}}$ is a valid RLWE encryption of the polynomial

$$\mathsf{DB}(X) \cdot X^{-d} = \sum_i \alpha_i X^{d_i - d} \in \mathcal{R}_t.$$

Since we use the cyclotomic polynomial $\phi_M(X) = X^N + 1$ of power-of-two degree, the polynomial ring $\mathcal{R}$ has the property $X^N = -1$. Thus, for any tag d, the constant term of the polynomial $\mathsf{DB}(X) \cdot X^{-d}$ is α_i if there is some index i satisfying $d = d_i$, otherwise zero.

Now the server applies the RLWE.Conv algorithm on $\mathsf{ct}_{\mathsf{mult}}$ to compute an LWE encryption $\mathsf{ct}_{\mathsf{conv}}$ of this constant term. This conversion procedure not only prevents the leakage of information that has not been queried but also reduces the size of output ciphertext by half. In addition, the (optional) modulus-switching procedure can be considered to get a ciphertext $\mathsf{ct}_{\mathsf{res}}$ with a smaller modulus size and reduce the communication cost. Finally the user decrypts this LWE ciphertext and gets the desired value α_i or zero ($\perp$). Algorithm 1 summarizes the procedure of secure *search-and-extraction*.

Our method can be modified to support a secure comparison of data values using a hash (one-way) function. If hashed values of α_i are used as polynomial coefficients, our method will return a hashed value of α_i to the user instead of α_i. The user may check whether the resulting value and the hashed query value are the same or not without knowing information about database.

Algorithm 1 Procedure of secure search-and-extraction

```
1: Database encryption: The data owner encodes the
   genomic information as $\mathsf{DB}(X)$ and submits its
   encryption to the server:
       $\mathsf{ct}_{\mathsf{DB}} \leftarrow \mathsf{RLWE.Enc}(\mathsf{DB}(X))$.
2: Query encryption: The user encodes the query tag $d$
   and sends its encryption to the server:
       $\mathsf{CT}_{\mathsf{Q}} \leftarrow \mathsf{RGSW.Enc}\left(X^{-d}\right)$.
3: Evaluation phase: The server computes their multipli-
   cation, and carries out the conversion and modulus-
   switching operations:
       $\mathsf{ct}_{\mathsf{mult}} \leftarrow \mathsf{Hybrid.Mult}(\mathsf{CT}_{\mathsf{Q}}, \mathsf{ct}_{\mathsf{DB}})$.
       $\mathsf{ct}_{\mathsf{conv}} \leftarrow \mathsf{RLWE.Convert}(\mathsf{ct}_{\mathsf{mult}})$.
       $\mathsf{ct}_{\mathsf{res}} \leftarrow \mathsf{LWE.ModSwitch}(\mathsf{ct}_{\mathsf{conv}})$.
   Return the resulting ciphertext $\mathsf{ct}_{\mathsf{res}}$ to the user.
4: Decryption phase: The user decrypts the ciphertext
   with the secret key and gets the desired value:
       $\alpha \leftarrow \mathsf{LWE.Dec}(\mathsf{ct}_{\mathsf{res}})$.
```

Comparison with related work

Equality test has been traditionally considered difficult to perform on homomorphic encryption, because of its large circuit depth [7, 15, 16]. They evaluate the equality test on each encrypted tuple of database, so at least $\Omega(n)$ homomorphic operations are required for searching on database of size n. In addition, Boneh et al. [17] does not protect the database information to the users, that is, the whole database can be recovered by the resulting ciphertext of a query. However, our method is very efficient in parameter size and complexity since it requires only a single hybrid multiplication.

One limitation of this method is that the tags d_i should be bounded by ciphertext dimension N to construct the encoding polynomial $\mathsf{DB}(X)$. Since the dimension N has a significant influence on the performance of HE scheme, too large value of N has an impractical impact on the performance. In the next section, we will describe how to overcome this problem in terms of the application to genomic data.

Secure searching of biomarkers

We return to our main goal of task3: secure outsourcing matching of a set of biomakers to encrypted genomes. We describe how to encode and encrypt the genotype information of VCF file in order to apply the privacy-preserving database searching and extraction.

VCF file contains multiple genotype information lines, where each of them consists of a triple $(\mathsf{ch}_i, \mathsf{pos}_i, \mathsf{SNPs}_i)$ of chromosome number, position, and a sequence of SNP alleles. A chromosome identifier ch ranges from 1 to 22, X, and Y. A non-negative integer pos represents the reference position with the first base having position 1, and SNPs is a r eference or alternate sequence in $\{A, T, G, C\}^*$. A query from user is also a triple of the same form and we aim to decide absence/presence of this biomarker in the database file.

We represent the sex chromosomes X and Y as 0 and 23, respectively. Then we define an encoding function $\mathcal{E} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ by

$$(\mathsf{ch}, \mathsf{pos}) \mapsto d = \mathsf{ch} + 24 \cdot \mathsf{pos}.$$

In the following, we describe how to encode the SNPs. For convenience we set the upper bound for the length of SNPs, so let n_{SNP} be the maximal number of reference (or alternate) alleles to be compared between the query genome and user genome in the target database. Each of SNP is represented by two bits as

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

and then concatenated with each other. Next we pad with 1 to the left of the bit string in order to express the staring position of SNPs. Finally it is zero-padded into a binary string of length $\ell_{\mathsf{SNP}} = 2 \cdot n_{\mathsf{SNP}} + 1$, and we convert it into an integer value, denoted by α_i. If a single nucleotide variant at the given locus is not known, then it is encoded as 0-string. For example, '*GC*' is encoded as a bit string 1|10|11, which will be represented as an integer $1|10|11_{(2)} = 27$.

Now consider the case that we wish to encode the reference and alternate alleles together. Let α_i^{ref} and α_i^{alt} denote the integer encodings of n_{SNP} reference alleles and n_{SNP} alternate alleles, respectively. Then we define an encoding α_i by the concatenation of two encodings, *i.e.*, $\alpha_i = 2^{\ell_{\mathsf{SNP}}} \cdot \alpha_i^{\mathsf{ref}} + \alpha_i^{\mathsf{alt}}$ as an integer. Table 1 shows the format of database file and illustrates some examples of encoded genomic data.

A database file is encoded as a set of pair (d_i, α_i) for $i = 1, \ldots, n$ such that $d_i = \mathcal{E}(\mathsf{ch}_i, \mathsf{pos}_i)$ and α_i is the encoded integer of the i-th SNP allele string. Then the encodings d_i and α_i are regarded as data-tag and value attribute, respectively. The data user constructs a polynomial $\mathsf{DB}(X) = \sum_k c_k X^k$ such that

$$c_k = \begin{cases} \alpha_i & \text{if } k = d_i \text{ for some } i, \\ \alpha \leftarrow \mathbb{Z}_t & \text{otherwise.} \end{cases}$$

The user encrypts the polynomial with the RLWE public-key encryption scheme as described above.

The query genes are also encoded as a pair of integers (d, α), however, we consider only the information of d is encrypted using the RGSW symmetric encryption scheme, that is, the user encrypts the monomial X^{-d}.

Results and discussion

In this section, we explain how to set the parameters and describe our optimization techniques for the implementation. We also present our results using the techniques. The dataset was randomly selected from Personal Genome Project. Our implementation is publicly available on github [18].

How to set parameters

Since all the matching computation is performed on encrypted data in the cloud, the security against a semi-honest adversary follows from the semantic security of the underlying HE scheme. The security of the homomorphic encryption scheme relies on the hardness of the RLWE assumption. We derive a lower-bound on the ring dimension as $N \geq \frac{\lambda+110}{7.2} \cdot \log_2 Q$ to get λ-bit security level from the security analysis of [11].

Table 1 The format of genome data and its encoding with $n_{\mathsf{SNP}} = 10$

CHROM	POS	d	REF	ALT	α
1	161235340	3869648161	*G*	*A*	12582916
1	161235596	3869654305	*C*	*T*	14680069
1	161235657	3869655769	*G*	*T*	12582917
1	161235981	3869663545	*G*	*A*	12582916
1	161237503	3869700073	·	*TTTTTGT*	21849
1	161237891	3869709385	*G*	*A*	12582916
1	161238009	3869712217	*G*	·	12582912
1	161238488	3869723713	*A*	*G*	8388614
1	161238683	3869728393	*G*	*A*	12582916
1	161238856	3869732545	*T*	·	10485760
1	161239028	3869736673	*AG*	·	37748736
1	161239142	3869739409	*A*	*G*	8388614
1	161239346	3869744305	*G*	*T*	12582917
1	161239470	3869747281	*C*	*T*	14680069
1	161239788	3869754913	·	*AA*	16
1	161239978	3869759473	*C*	*T*	12582917
1	161240641	3869775385	*TGAT*	·	740294656

Given the ciphertext modulus Q, it follows from the estimation of noise growth during evaluations [12] and decryption condition that we get the upper bound on the plaintext modulus t to ensure the correctness of decryption after computation. So we set t as the largest power-of-two integer less than the upper bound. If the encodings of the allele strings are too large, we divide them into smaller integers so that each of them is smaller than t. Then we repeat the algorithm to construct the corresponding polynomials of each integer.

Optimization techniques

As we mentioned before, the ring dimension N needs to be larger than the encoded integers d_i's. However, the encoded integers d_i from VCF files have bits size about 32, while a dimension N with about $11 \le \log_2 N \le 16$ is considered appropriate for implementation of HE schemes to achieve both security and efficiency. Hence direct application of our method to the VCF file would yield an impractical result.

For compression of tag data and its re-randomization, we make the use of a pseudo random number generator $H(\cdot)$ which transforms a tag d_i into a pair of two non-negative integers d_i^* and $d_i^\dagger$ less than N. Our implementation adopts SHA-3 and extracts $\log_2 N = 11$ bits of the hashed value for each of d_i^* and $d_i^\dagger$.

We construct two polynomials

$$\mathsf{DB}^*(X) = \sum_k c_k^* X^k,\ \mathsf{DB}^\dagger(X) = \sum_k c_k^\dagger X^k$$

by the Algorithm 2. Note that for any $1 \le i \le n$ and $H(d_i) = (d_i^*, d_i^\dagger) \in \{0, \ldots, N-1\}^2$, the pair of constructed polynomials DB^* and $\mathsf{DB}^\dagger$ satisfy $\alpha_i = c_{d_i^*} + c_{d_i^\dagger}$. The procedure of database encoding for secure search of biomarkers is described in Algorithm 2.

Algorithm 2 Encoding genomic data

```
1:  $c^*_{d_1^*} \leftarrow \alpha_1 \in \mathbb{Z}_q, c^\dagger_{d_1^\dagger} \leftarrow \alpha_1 - c^*_{d_1^*}$
2:  $d_1^* \in \mathcal{D}^*, d_1^\dagger \in \mathcal{D}^\dagger$
3:  for $i \in \{2, \ldots, n\}$ do
4:      if $d_i^* \notin \mathcal{D}^*$ and $d_i^\dagger \notin \mathcal{D}^\dagger$ then
5:          $c^*_{d_i^*} \leftarrow \alpha_i \in \mathbb{Z}_q, c^\dagger_{d_i^\dagger} \leftarrow \alpha_i - c^*_{d_i^*}$
6:      else if $d_i^* \in \mathcal{D}^*$ and $d_i^\dagger \notin \mathcal{D}^\dagger$ then
7:          $c^\dagger_{d_i^\dagger} \leftarrow \alpha_i - c^*_{d_i^*}$
8:      else if $d_i^* \notin \mathcal{D}^*$ and $d_i^\dagger \in \mathcal{D}^\dagger$ then
9:          $c^*_{d_i^*} \leftarrow \alpha_i - c^\dagger_{d_i^\dagger}$
10:     end if
11:     $d_i^* \in \mathcal{D}^*, d_i^\dagger \in \mathcal{D}^\dagger$
12: end for
13: return $\mathsf{DB}^*(X) = \sum_k c_k^* X^k, \mathsf{DB}^\dagger(X) = \sum_k c_k^\dagger X^k$
```

Let $\mathsf{ct}^*_{\mathsf{DB}}$ and $\mathsf{ct}^\dagger_{\mathsf{DB}}$ denote the ciphertexts of the polynomials DB^* and $\mathsf{DB}^\dagger$, respectively. Similarly, given the query encoding d, the user computes its randomized value $H(d) = (d^*, d^\dagger)$ and encrypts the two polynomials X^{-d^*} and $X^{-d^\dagger}$. We denote the ciphertexts by $\mathsf{CT}^*_{\mathsf{Q}}$ and $\mathsf{CT}^\dagger_{\mathsf{Q}}$. The server computes the hybrid multiplication to obtain the ciphertexts

$$\mathsf{ct}^*_{\mathsf{mult}} = \mathsf{CT}^*_{\mathsf{Q}} \boxdot \mathsf{ct}^*_{\mathsf{DB}},\ \mathsf{ct}^\dagger_{\mathsf{mult}} = \mathsf{CT}^\dagger_{\mathsf{Q}} \boxdot \mathsf{ct}^\dagger_{\mathsf{DB}}.$$

Now let ct denote the ciphertext computed by the homomorphic addition between $\mathsf{ct}^*_{\mathsf{mult}}$ and $\mathsf{ct}^\dagger_{\mathsf{mult}}$. Finally the server converts it into an LWE ciphertext and performs the modulus-switching procedure as described above. The Algorithm 3 describes the procedure of secure *search-and-extraction* using our proposed optimization techniques.

Implementation results

The use of variable type '*int32_t*' accelerates the speed of implementations and basic C++ std libraries, so we set $Q = 2^{32}$ as the ciphertext modulus. We also set $t = 2^{11}$ as the modulus parameter of the plaintext space to ensure

Algorithm 3 Procedure of optimized secure search of biomarkers

```
1: Database encryption: The data owner encodes the
   genomic information as $\mathsf{DB}^*(X)$ and $\mathsf{DB}^\dagger(X)$. Then the
   user submits the ciphertexts to the server:
       $\mathsf{ct}^*_{\mathsf{DB}} \leftarrow \mathsf{RLWE.Enc}(\mathsf{DB}^*(X))$,
       $\mathsf{ct}^\dagger_{\mathsf{DB}} \leftarrow \mathsf{RLWE.Enc}(\mathsf{DB}^\dagger(X))$.
2: Query encryption: The user encodes the query as
   $X^{-d^*}$ and $X^{-d^\dagger}$. Then the user sends the ciphertexts
   to the server:
       $\mathsf{CT}^*_{\mathsf{Q}} \leftarrow \mathsf{RGSW.Enc}(X^{-d^*})$,
       $\mathsf{CT}^\dagger_{\mathsf{Q}} \leftarrow \mathsf{RGSW.Enc}(X^{-d^\dagger})$.
3: Evaluation phase: The server computes their multipli-
   cations:
       $\mathsf{ct}^*_{\mathsf{mult}} \leftarrow \mathsf{CT}^*_{\mathsf{Q}} \boxdot \mathsf{ct}^*_{\mathsf{DB}}, \mathsf{ct}^\dagger_{\mathsf{mult}} \leftarrow \mathsf{CT}^\dagger_{\mathsf{Q}} \boxdot \mathsf{ct}^\dagger_{\mathsf{DB}}$.
   Let $\mathsf{ct} \leftarrow \mathsf{ct}^*_{\mathsf{mult}} + \mathsf{ct}^\dagger_{\mathsf{mult}}$. The server converts it into
   an LWE ciphertext and performs modulus-switching
   operations:
       $\mathsf{ct}_{\mathsf{conv}} \leftarrow \mathsf{RLWE.Convert}(\mathsf{ct})$.
       $\mathsf{ct}_{\mathsf{res}} \leftarrow \mathsf{LWE.ModSwitch}(\mathsf{ct}_{\mathsf{conv}})$.
   Return the resulting ciphertext $\mathsf{ct}_{\mathsf{res}}$ to the user.
4: Decryption phase: The user decrypts the ciphertext
   with the secret key and gets the desired value:
       $\alpha \leftarrow \mathsf{LWE.Dec}(\mathsf{ct}_{\mathsf{res}})$.
```

Table 2 Implementation results of secure searching of biomarkers

DB size	n_{SNP}	Complexity				Storage		
		Query-enc	DB-enc	Eval	Dec	Query	DB	Result
10 K	2	3.247 ms	3.563 ms	0.018 s	0.004 ms	160 KB	3 MB	0.75 MB
	5		7.212 ms	0.039 s	0.011 ms		6 MB	1.5 MB
	10		14.813 ms	0.079 s	0.027 ms		12 MB	3 MB
100 K	2		21.424 ms	0.111 s	0.034 ms		17 MB	4.25 MB
	5		42.415 ms	0.227 s	0.064 ms		34 MB	8.5 MB
	10		99.921 ms	0.454 s	0.139 ms		68 MB	17 MB
4 M	2		0.745 s	3.954 s	1.171 ms		593 MB	148 MB
	5		1.506 s	7.911 s	1.949 ms		1185 MB	296 MB
	10		3.001 s	15.442 s	3.795 ms		2370 MB	593 MB

the correctness for the output ciphertext. We take the following parameters for Gadget matrix $\mathbf{G}$: $B_{\mathsf{g}} = 128$ and $d_{\mathsf{g}} = 5$, so that they satisfy the condition $B_{\mathsf{g}}^{d_{\mathsf{g}}} \geq Q$.

Each coefficient of the secret key sk is chosen at random from $\{0, \pm 1\}$ and we set 64 as the number of nonzero coefficients in the secret key. As in the work of [12], we considered the Gaussian distribution of standard deviation $\sigma = 1.4$ to sample random error polynomials.

For the efficiency of homomorphic multiplication, we also used the optimized library for complex FFT, *i.e., the Fast Fourier Transform in the West* [19]. That is, we use the complex primitive $2N$-th root of unity rather than a primitive root in a prime field of order Q. We measure a running time of 0.804 s to set up the FFT environment at dimension $2N = 2^{12}$. The key generation of two schemes takes about 0.247 ms in total.

Table 2 presents the time complexity and storage for the evaluation of secure searching of biomarkers. All the experiments were performed on a single Intel Core i5 running at 2.9 GHz processor. The chosen parameters provide $\lambda = 128$ bits of security level.

Conclusions

In this work, we suggested an efficient method to securely search the query tag and extract the corresponding value from a database over hybrid GSW homomorphic encryption scheme. We came up with a solution to the secure outsourcing matching problem by using polynomial encoding and extraction of desired value based on the multiplication of an RGSW ciphertext and an ordinary RLWE ciphertext. And then we applied this method to find a set of biomarkers in DNA sequences.

Our solution shows the progress of cryptographic techniques in terms of their capability can support real-world genome data analysis in a cloud environment. We list a few fascinating open problems to remain. First, we only considered the semi-honest adversary model in this work. Other tools such as homomorphic authenticated scheme may lead to more efficient protocols in the malicious settings. Another issue is to support k multiple queries while maintaining the performance and communication cost less than k times of a single query case. We expect to have much faster performance by enabling a batching method.

Acknowledgements
The authors would like to thank the referee for helpful comments. The authors would also like to thank the iDASH Secure Genome Analysis Contest organizers, in particular Xiaoqian Jiang and Shuang Wang, for running the contest and providing the opportunity to submit competing implementations for these important tasks.

Funding
Publication of this article has been funded by IT R&D program of MSIP/KEIT (No. B0717-16-0098).

Availability of data and materials
Not applicable.

Authors' contributions
MK, YS, and JC designed the baseline methods. MK and YS drafted the manuscript and conducted the experiment for the competition. JC guided the experimental design and provided detailed edits. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

About this supplement
This article has been published as part of BMC Medical Genomics Volume 10 Supplement 2, 2017: Proceedings of the 5th iDASH Privacy and Security Workshop 2016. The full contents of the supplement are available online at https://bmcmedgenomics.biomedcentral.com/articles/supplements/volume-10-supplement-2.

Author details
[1]Division of Biomedical Informatics, University of California- San Diego, San Diego, CA 92093, USA. [2]Department of Mathematical Sciences, Seoul National University, GwanAkRo 1, Seoul 08826, Republic of Korea.

Published: 26 July 2017

References

1. Humbert M, Ayday E, Hubaux J-P, Telenti A. Addressing the concerns of the lacks family: quantification of kin genomic privacy. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. ACM; 2013. p. 1141–52.
2. Erlich Y, Narayanan A. Routes for breaching and protecting genetic privacy. Nat Rev Genet. 2014;15(6):409–21.
3. Naveed M, Ayday E, Clayton EW, Fellay J, Gunter CA, Hubaux J-P, Malin BA, Wang X. Privacy in the genomic era. ACM Comput Surv (CSUR). 2015;48(1):6.
4. Yasuda M, Shimoyama T, Kogure J, Yokoyama K, Koshiba T. Secure pattern matching using somewhat homomorphic encryption. In: Proceedings of the 2013 ACM Cloud Computing Security Workshop. ACM; 2013. p. 65–76.
5. Lauter K, López-Alt A, Naehrig M. Private computation on encrypted genomic data. In: International Conference on Cryptology and Information Security in Latin America. Springer International Publishing; 2014. p. 3–27.
6. Cheon JH, Kim M, Lauter K. Homomorphic computation of edit distance. In: International Conference on Financial Cryptography and Data Security. Springer Berlin Heidelberg; 2015. p. 194–212.
7. Kim M, Lauter K. Private genome analysis through homomorphic encryption. BMC Med Inform Decis Mak. 2015;15(Suppl 5):3.
8. Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory. 2014;6(3):13.
9. Brakerski Z. Fully homomorphic encryption without modulus switching from classical gapsvp. In: Advances in Cryptology-CRYPTO. Springer Berlin Heidelberg; 2012. p. 868–86.
10. Lyubashevsky V, Peikert C, Regev O. On ideal lattices and learning with errors over rings. In: Advances in Cryptology-EUROCRYPT. Springer Berlin Heidelberg; 2010. p. 1–23.
11. Gentry C, Halevi S, Smart NP. Homomorphic evaluation of the AES circuit. In: Advances in Cryptology-CRYPTO. Springer Berlin Heidelberg; 2012. p. 850–67.
12. Ducas L, Micciancio D. Fhew: Bootstrapping homomorphic encryption in less than a second. In: Advances in Cryptology-EUROCRYPT. Springer Berlin Heidelberg; 2015. p. 617–40.
13. Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology-CRYPTO. Springer Berlin Heidelberg; 2013. p. 75–92.
14. Chillotti I, Gama N, Georgieva M, Izabachene M. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 s. In: Advances in Cryptology-ASIACRYPT. Springer Berlin Heidelberg; 2016. p. 3–33.
15. Cheon JH, Kim M, Kim M. Search-and-compute on encrypted data. In: International Conference on Financial Cryptography and Data Security. Springer Berlin Heidelberg; 2015. p. 142–59.
16. Cheon JH, Kim M, Kim M. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Trans Inf Forensic Secur. 2016;11(1):188–99.
17. Boneh D, Gentry C, Halevi S, Wang F, Wu DJ. Private database queries using somewhat homomorphic encryption. In: International Conference on Applied Cryptography and Network Security. Springer Berlin Heidelberg; 2013. p. 102–18.
18. Kim M, Song Y. Implementation of Secure Searching of Biomarkers. 2016. http://github.com/amedonis/HybridHE.
19. Frigo M, Johnson SG. The design and implementation of fftw3. Proc IEEE. 2005;93(2):216–31.