Private Genome Analysis through Homomorphic Encryption

Miran Kim[1] and Kristin Lauter[2]

[1] Seoul National University (SNU), Republic of Korea
alfks500@snu.ac.kr
[2] Cryptography Research Group, Microsoft Research
klauter@microsoft.com

Abstract. The rapid development of genome sequencing technology allows researchers to access large genome datasets. However, outsourcing the data processing to the cloud poses high risks for personal privacy. The aim of this paper is to give a practical solution for this problem using homomorphic encryption. In our approach, all the computations can be performed in an untrusted cloud without requiring the decryption key or any interaction with the data owner, which preserves the privacy of genome data.

In this paper, we present evaluation algorithms for secure computation of the minor allele frequencies and χ^2 statistic in a genome-wide association studies setting. We also describe how to privately compute the Hamming distance and approximate Edit distance between encrypted DNA sequences. Finally, we compare performance details of using two practical homomorphic encryption schemes - the BGV scheme by Gentry, Halevi and Smart and the YASHE scheme by Bos, Lauter, Loftus and Naehrig. Such an approach with the YASHE scheme analyzes data from 400 people within about 2 seconds and picks a variant associated with disease from 311 spots. For another task, using the BGV scheme, it took about 65 seconds to securely compute the approximate Edit distance for DNA sequences of size 5K and figure out the differences between them.

Keywords: Homomorphic encryption, Genome-wide association studies, Hamming distance, Approximate Edit distance.

1 Introduction

The rapid development of genome sequencing technology has led to the genome era. We expect that the price of a whole genome sequence will soon be $1K in a day, which enables researchers to access large genome datasets. Moreover, many genome projects like the Personal Genome Project (PGP) [2] and the HapMap Project [1] display genotypic information in public databases, so genomic data has become publicly accessible.

Privacy Threats from Exposing Genomic Data. While genome data can be used for a wide range of applications including healthcare, biomedical research, and forensics, it can be misused, violating personal privacy via genetic disease disclosure or genetic discrimination. Even when explicit identifiers (*e.g.,* name, date of birth or address) are removed from genomic data, one can often recover the identity information [18, 12, 26]. For these reasons, genomic data should be handled with care.

Privacy through Encryption. There have been many attempts to protect genomic privacy using cryptographic methods. In particular, it has been suggested that we can preserve privacy through (partially) homomorphic encryption, which allows computations to be carried out on ciphertexts. Kantarcioglu et al. [19] presented a novel framework that allows organizations to support data mining without violating genomic privacy. Baldi et al. [4] proposed a cryptographic protocol to determine whether there exists a biological parent-child relationship between two individuals. Ayday et al. [3] recently conducted privacy-preserving computation of disease risk based on genomic and non-genomic data. However, these methods used homomorphic computation involving a single operation on ciphertexts (*e.g.,* either additions or multiplications, not both), thus they could support a limited set of genomic queries.

Fully homomorphic encryption (*e.g.,* [13, 30, 15]) permits encrypted data to be computed on without decryption, so it allows us to evaluate arbitrary arithmetic circuits over encrypted data. Thus, we can privately perform all types of genome analysis using Homomorphic Encryption (HE) cryptosystems. Moreover, we can delegate intensive computation to a public cloud and store large amounts of data in it.

Recently, many protocols to conduct privacy-preserving computation of genomic tests with fully homomorphic encryption have been introduced. Yasuda et al. [32] gave a practical solution for computation of multiple Hamming distance values using the LNV scheme [21] on encrypted data, so to find the locations where a pattern occurs in a text. Graepel et al. [16] and Bos et al. [7] applied HE to machine learning, and described how to privately conduct predictive analysis based on an encrypted learned model. Lauter et al. [20] gave a solution to privately compute the basic genomic algorithms used in genetic association studies. Cheon et al. [11] described how to calculate edit distance on homomorphically encrypted data.

Scenarios. One possible scenario could be of interest in situations involving patients, a data owner (*e.g.,* a healthcare organization or a medical center) and a public cloud. In our solution, a data owner wants to store large amounts of data in the cloud and many users may interact with the same data over time. The cloud can handle all that interaction through computation on encrypted data, so it does not require further interaction from the data owner. The patients can upload their encrypted data directly to the cloud using the public key. The genomic tests are performed on the cloud and the encrypted results are returned to the data owner. Finally, the data owner decrypts the results using the secret key to share it with the patient. All the computations in the cloud are performed on encrypted data without requiring the decryption key, so the privacy of genomic data can be protected by the semantic security of the underlying HE schemes.

Our Contributions. In this paper, we propose efficient evaluation algorithms to compute genomic tests on encrypted data. We first consider the basic tests which are used in Genome-Wide Association Studies (GWAS). They are conducted to analyze the statistical associations between common genetic variants in different individuals. In particular, we focus on the minor allele frequencies (MAFs) and χ^2 test statistic between the variants of case and control groups. Secondly, we consider DNA sequence comparison which can be used in sequence alignment and gene finding. We show how to privately compute the Hamming distance and approximate Edit distance on encrypted data. We also adapt these methods to the practical HE schemes – BGV scheme [14] by Gentry, Halevi and Smart and YASHE scheme [6] by Bos, Lauter, Loftus and Naehrig. Finally, we compare the performance of the two encryption schemes in these contexts. In practice, we take advantage of batching techniques to parallelize both space and computation time together.

2 Background on Genome Analysis

The iDASH (Integrating Data for Analysis, 'anonymization' and SHaring) National Center organized the iDASH Privacy & Security challenge for secure genome analysis. This paper is based on a submission to the iDASH challenge which consisted of two tasks: i) secure outsourcing of GWAS and ii) secure comparison between genomic data.

2.1 Task 1: GWAS on Genomic Data

Given the encrypted genotypes of two groups of individuals over many single nucleotide variants (SNVs), the goal of this task is to privately compute the MAFs in each group and a χ^2 test statistic between the two groups on each site.

Suppose that A and B are two alleles of the gene, and let n_{AA}, n_{AB}, n_{BB} denote the numbers of observed individuals for genotypes AA, AB, BB, respectively. The allele counts of A and B are given by $n_A \overset{let}{=} 2n_{AA} + n_{AB}$ and $n_B \overset{let}{=} 2n_{BB} + n_{AB}$. Then the MAF of the given alleles is defined by

$$\frac{\min(n_A, n_B)}{n_A + n_B}.$$

If we let N be the total number of people in a sample population, the total number of alleles in the sample is $n_A + n_B = 2N$, so we compute only one of two allele counts in encrypted form. The minimum can then easily be computed after decryption and we obtain the MAF by one division by $2N$.

The χ^2 test statistic in case-control groups is computed based on the allelic contingency table (Table 1):

$$\frac{T(n_A n'_B - n_B n'_A)^2}{R \cdot S \cdot G \cdot K}.$$

We observe that the test can be written as a function of n_A and n'_A. More precisely, it is expressed as

$$\begin{aligned} & \frac{4N\left(n_A(2N - n'_A) - n'_A(2N - n_A)\right)^2}{2N \cdot 2N \cdot G \cdot K} \\ = & \frac{4N\left(n_A - n'_A\right)^2}{\left(n_A + n'_A\right) \cdot \left(4N - \left(n_A + n'_A\right)\right)}. \end{aligned}$$

Let $n_A^{(j)}$ and $n_A^{\prime(j)}$ denote the allele counts of A at SNV j in the case group and control group, respectively. As discussed above, it suffices to compute $(n_A^{(j)} + n_A^{\prime(j)})$ and $(n_A^{(j)} - n_A^{\prime(j)})$ over encrypted data.

	Allele type		Total
	A	B	
Case	n_A	n_B	$R = 2N$
Control	n'_A	n'_B	$S = 2N$
Total	$G = n_A + n'_A$	$K = n_B + n'_B$	$T = 4N$

Table 1. Allelic Contingency Table

2.2 Task 2: Human Genome Comparison

The goal of the second task is to privately compute the Hamming distance and approximate Edit distance between the encrypted genome sequences. Suppose that two participants have Variation Call Format (VCF) files which summarize their variants compared with the reference genome (*e.g.,* insertion, deletion, or substitution at a given position of a given chromosome).

Algorithm 1 Hamming Distance Algorithm

```
$h \leftarrow 0$
for $i \in \mathcal{L}$ do
    if (‘$x_i$.sv’ or ‘$y_i$.sv’) in $\{$‘INS’, ‘DEL’$\}$ then
        $h_i \leftarrow 0$
    else if (($x_i$ or $y_i$) == ‘$\emptyset$’) or
     (($x_i$.ref == $y_i$.ref) and ($x_i$.alt ! = $y_i$.alt)) then
        $h_i \leftarrow 1$
    else
        $h_i \leftarrow 0$
    end if
    $h \leftarrow h + h_i$
end for
return $h$
```

Algorithm 2 Approximate Edit Distance Algorithm

```
$e \leftarrow 0$
for $i \in \mathcal{L}$ do
    if $x_i$ == ‘$\emptyset$’ then
        $D(x_i) \leftarrow 0$
    else if ‘$x_i$.sv’ == ‘DEL’ then
        $D(x_i) \leftarrow$ len($x_i$.ref)
    else
        $D(x_i) \leftarrow$ len($x_i$.alt)
    end if
    Define $D(y_i)$ with the same way as $D(x_i)$
    if (($x_i$.ref == $y_i$.ref) and ($x_i$.alt == $y_i$.alt)) then
        $e_i \leftarrow 0$
    else
        $e_i \leftarrow \max\{D(x_i), D(y_i)\}$
    end if
    $e \leftarrow e + e_i$
end for
return $e$
```

If there is only one record in the VCF files at a specified location, the other one is considered to be an empty set (‘$\emptyset$’). Let $\mathcal{L}$ be a list indexed by the positions of two participants. Then we can define the Hamming distance as described in Algorithm 1, where “x_i.sv” denotes the type of structural variant relative to the reference, “x_i.ref ” the reference bases and “x_i.alt” the alternate non-reference alleles.

The standard dynamic programming approach to compute the full Wagner-Fischer Edit distance [31] is computed in a recursive way, so the multiplicative depth of the circuit to be homomorphically evaluated is too large. Recently, Cheon et al. [11] presented an algorithm to compute the WF Edit distance over packed ciphertexts but it took about 27 seconds even on length 8 DNA sequences. On the other hand, in this task we are given the distance to a public human DNA sequence (called the reference genome), which allows us to efficiently approximate the Edit distance using Algorithm 2. It is calculated based on the set difference metric, which enables parallel processing in computation.

3 Practical Homomorphic Encryptions

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 [9], 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 [8] such that the same modulus is used throughout the evaluation process.

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 $R = \mathbb{Z}[x]/(\Phi_m(x))$, set the plaintext space to $R_t := R/tR$ for some fixed $t \geq 2$ and the ciphertext space to $R_q := R/qR$ 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 $\mathcal{D}$. Now, we recall the BGV scheme [14] and the scale-invariant YASHE scheme [6].

3.1 The BGV scheme

Gentry, Halevi and Smart [14] constructed an efficient BGV-type SHE scheme. The security of this scheme is based on the (decisional) Ring Learning With Errors (RLWE) assumption, which was first introduced by Lyubashevsky, Peikert and Regev [25]. The assumption is that it is infeasible to distinguish the following two distributions. The first distribution consists of pairs (a_i, u_i), where $a_i, u_i \leftarrow R_q$ uniformly at random. The second distribution consists of pairs of the form (a_i, b_i)=$(a_i, a_i\mathsf{s} + e_i)$ where $a_i \leftarrow R_q$ drawn uniformly and $\mathsf{s}, e_i \leftarrow \chi$. Note that we can generate RLWE samples as $(a_i, a_i\mathsf{s} + te_i)$ where t and q are relatively prime. To improve efficiency for HE, they use very sparse secret keys s with coefficients sampled from $\{-1, 0, 1\}$. Here is the SHE scheme of [14]:

- $\mathsf{ParamsGen}$: Given the security parameter λ, choose an odd integer m, a chain of moduli $q_0 < q_1 < \cdots < q_{L-1} = q$, a plaintext modulus t with $1 < t < q_0$, and discrete Gaussian distribution χ_{err}. Output $(m, \{q_i\}, t, \chi_{err})$.
- KeyGen: On the input parameters, choose a random s from $\{0, \pm 1\}^{\phi(m)}$ and generate an RLWE instance $(a, b) = (a, [a\mathsf{s} + te]_q)$ for $e \leftarrow \chi_{err}$. For an integer P, we define the key switching matrix
$$W = \begin{pmatrix} b_{\mathsf{s}} \\ a_{\mathsf{s}} \end{pmatrix} \text{ where } b_{\mathsf{s}} = [a_{\mathsf{s}} \cdot \mathsf{s} + te_{\mathsf{s}} + P\mathsf{s}^2]_{Pq_{L-2}}$$
for $a_{\mathsf{s}} \leftarrow R_q$ uniformly at random and $e_{\mathsf{s}} \leftarrow \chi_{err}$. We set the key pair: $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) = ((a, b), \mathsf{s}, W)$. Then we define the $\mathsf{SwitchKey}(\mathsf{c}, \mathsf{evk})$ for the extended ciphertext $\mathsf{c} = (d_0, d_1, d_2)$ at level l as follows: set
$$\mathsf{c}' = \begin{pmatrix} c_0' \\ c_1' \end{pmatrix} = \left[\begin{pmatrix} Pd_0 \; [b_{\mathsf{s}}]_{Pq_l} \\ Pd_1 \; [a_{\mathsf{s}}]_{Pq_l} \end{pmatrix} \begin{pmatrix} 1 \\ d_2 \end{pmatrix} \right]_{Pq_l},$$
and then take an element $\mathsf{c}'' \in R_{q_l}$ such that $\mathsf{c}'' \equiv \mathsf{c}' (\text{mod } t)$ and c'' is the closet to $P \cdot \mathsf{c}'$ modulo t.

- Encryption: To encrypt $m \in R_t$, choose a small polynomial v and two Gaussian polynomials e_0, e_1 over R_q. Then compute the ciphertext given by

$$\mathsf{Enc}(m, \mathsf{pk}) = (c_0, c_1) = (m, 0) + (bv + te_0, av + te_1) \in R_q^2.$$

- Decryption: Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ at level l, output $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) = [c_0 - \mathsf{s} \cdot c_1]_{q_l} \bmod t$ where the polynomial $[c_0 - \mathsf{s} \cdot c_1]_{q_l}$ is called the *noise* in the ciphertext ct.
- Homomorphic Evaluation: Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{ct}' = (c_0', c_1')$ at level l, the homomorphic addition is computed by $\mathsf{ct}_{\mathsf{add}} = ([c_0 + c_0']_{q_l}, [c_1 + c_1']_{q_l})$. The homomorphic multiplication is computed by $\mathsf{ct}_{\mathsf{mult}} = \mathsf{SwitchKey}(c_0 * c_1, \mathsf{evk})$ where $c_0 * c_1 = ([c_0c_0']_{q_l}, [c_0c_1' + c_1c_0',]_{q_l}, [c_1c_1']_{q_l})$ and the key switching function SwitchKey is used to reduce the size of ciphertexts to two ring elements. We also apply modulus switching from q_i to q_{i-1} in order to reduce the noise. If we reach the smallest modulus q_0, we can no longer compute on ciphertexts.

Smart and Vercauteren [29] observed that R_t is isomorphic to $\prod_{i=1}^{\ell} \mathbb{Z}_t[x]/f_i(x)$ if $\Phi_m(x)$ factors modulo t into ℓ irreducible factors $f_i(x)$ of the same degree. Namely, a plaintext polynomial m can be considered as a vector of ℓ small polynomials, $m \bmod f_i$, called *plaintext slots*. We can also transform the plaintext vector $(m_1, \ldots, m_r) \in \prod_{i=1}^{\ell} \mathbb{Z}_t[x]/f_i(x)$ to an element $m \in R_t$ using the polynomial Chinese Remainder Theorem (*i.e., $m = \mathsf{CRT}(m_1, \ldots, m_r)$*). In particular, it is possible to add and multiply on the slots: if $m, m' \in R_t$ encode $(m_1, \ldots, m_\ell)$ and $(m_1', \ldots, m_\ell')$ respectively, then we see that $m + m' = m_i + m_i' \mod f_i$ and $m \cdot m' = m_i \cdot m_i' \mod f_i$. This technique was adapted to the BGV scheme.

3.2 The YASHE scheme

A practical SHE scheme, YASHE, was proposed in [6] based on combining ideas from [8, ?,24]. The security of this scheme is based on the hardness of the RLWE assumption similar to the one for BGV. It also relies on the Decisional Small Polynomial Ratio (DSPR) assumption which was introduced by Lopez-Alt, Tromer, and Vaikuntanathan [24]. Let $t \in R_q^\times$ be invertible in R_q, $y_i \in R_q$ and $z_i = y_i/t \pmod q$ for $i = 1, 2$. For $z \in R_q$, we define $\chi_z = \chi + z$ to be the distribution shifted by z. The assumption is that it is hard to distinguish elements of the form $h = a/b$, where $a \leftarrow y_1 + t\chi_{z_1}, b \leftarrow y_2 + t\chi_{z_2}$, from elements drawn uniformly from R_q. The YASHE scheme consists of the following algorithms.

- ParamsGen: Given the security parameter λ, choose m to be a power of 2 (the m-th cyclotomic polynomial is $\Phi_m(x) = x^n + 1$ $(n = \phi(m) = m/2)$, modulus q and t with $1 < t < q$, truncated discrete Gaussian distribution χ_{err} on R such that the coefficients of the polynomial are selected in the range $[-B(\lambda), B(\lambda)]$), and an integer base $\omega > 1$. Output $(m, q, t, \chi_{err}, \omega)$.
- KeyGen: On the input parameters, sample $f', g \leftarrow \{0, \pm 1\}^{\phi(m)}$ and set $f = [tf' + 1]_q$. If f is not invertible modulo q, choose a new f' and compute the inverse $f^{-1} \in R$ of f modulo q and set $h = [tgf^{-1}]_q$. Let $\ell_{\omega,q} = \lfloor \log_\omega(q) \rfloor + 1$ and define

$$\mathsf{D}_{\omega,q}(a) = ([a_i]_\omega)_{i=0}^{\ell_{\omega,q}-1}, \quad \mathsf{P}_{\omega,q}(a) = ([a\omega^i]_q)_{i=0}^{\ell_{\omega,q}-1},$$

where $a = \sum_{i=0}^{\ell_{\omega,q}-1} a_i\omega^i$, $a_i \in R$ with coefficients in $(-\omega/2, \omega/2]$. Sample $\mathbf{e}, \mathbf{s} \leftarrow \chi_{err}^{\ell_{\omega,q}}$ and compute $\gamma = [\mathsf{P}_{\omega,q}(f) + \mathbf{e} + h\mathbf{s}]_q \in R_q^{\ell_{\omega,q}}$. Then we set the key pair: $(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) = (h, f, \gamma)$. For a ciphertext ct, we define the $\mathsf{SwitchKey}(\mathsf{ct}, \mathsf{evk})$ by computing $\left[\langle \mathsf{D}_{\omega,q}(\mathsf{ct}), \mathsf{evk} \rangle\right]_q$. Note that the key switching function SwitchKey is used to transform a ciphertext decryptable under the original secret key f.

- Encryption: To encrypt $m \in R_t$, choose $e, s \leftarrow \chi_{err}$ and then compute the ciphertext

$$\mathsf{Enc}(m, \mathsf{pk}) = \left[\left\lfloor \frac{q}{t} \right\rfloor \cdot [m]_t + e + hs \right]_q \in R_q.$$

- Decryption: Given a ciphertext ct, output $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) = \left\lfloor \frac{t}{q} \cdot [f \cdot \mathsf{ct}]_q \right\rceil \bmod t$.
- Homomorphic Evaluation: Given two ciphertext ct and ct', the homomorphic addition is computed by $\mathsf{ct}_{\mathsf{add}} = [\mathsf{ct} + \mathsf{ct}']_q$. The homomorphic multiplication is computed by $\mathsf{ct}_{\mathsf{mult}} = \mathsf{SwitchKey}(\left[\left\lfloor \frac{t}{q} \mathsf{ct} \cdot \mathsf{ct}' \right\rceil\right]_q, \mathsf{evk})$ (see [6] for details).

4 Our Method for Private Genome Analysis

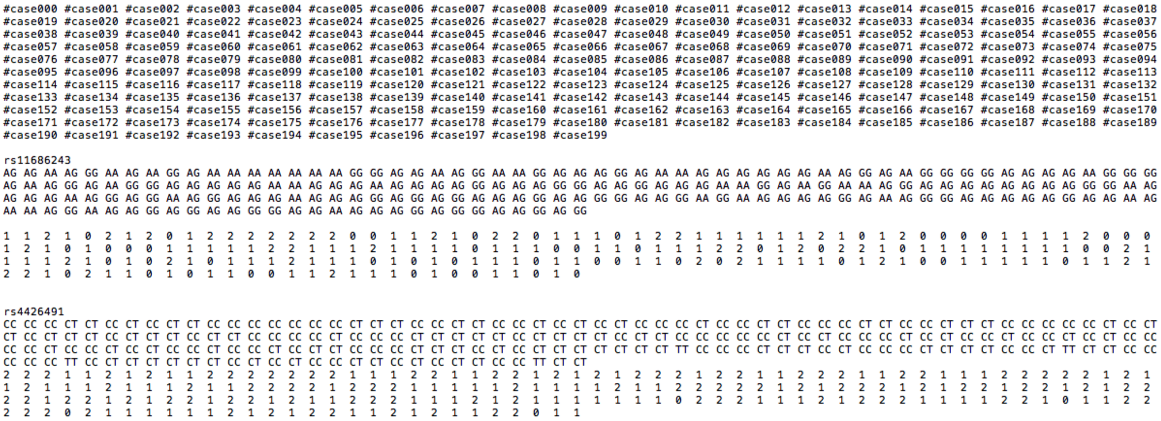
In this section, we describe how to encode and encrypt the genomic data for each task. Based on these methods, we propose the evaluation algorithms to compute the genomic tests on encrypted data.

4.1 Encoding Genomic Data

Lauter et al. [20] presented a method to encode a person's genotype given a candidate allele associated to a specified disease. They used a binary dummy vector representation, which makes the number of ciphertexts too large. In contrast, we encode the genotypes as integers so that one can efficiently compute their sums and differences over the integers. More precisely, for a bi-allelic gene with alleles A and B, there are 3 possible Single Nucleotide Polymorphisms (SNPs) - AA, AB, BB, and they are encoded as follows: $AA \to 2,\ AB \to 1,\ BB \to 0$. Figure 1 shows the file format of the data for task 1 and its encodings.

Now, we describe how genomic data can be encoded for DNA comparison. The first step is to curate the data using the positions in the VCF files of two participants. In other words, the server should arrange the information and make the merged list $\mathcal{L}$ so that each individual can encode their genotypic information according to the list. Let $\ell(\mathcal{L})$ denote the length of the list $\mathcal{L}$. Then, for $1 \le i \le \ell(\mathcal{L})$, we define two values

$$e_i = \begin{cases} 1 & \text{if } \mathrm{pos}_i \in \mathcal{L}, \\ 0 & \text{o.w,} \end{cases} \quad f_i = \begin{cases} 0 & \text{if } \mathrm{sv}_i \in \{\mathsf{INS}, \mathsf{DEL}\}, \\ 1 & \text{o.w,.} \end{cases}$$



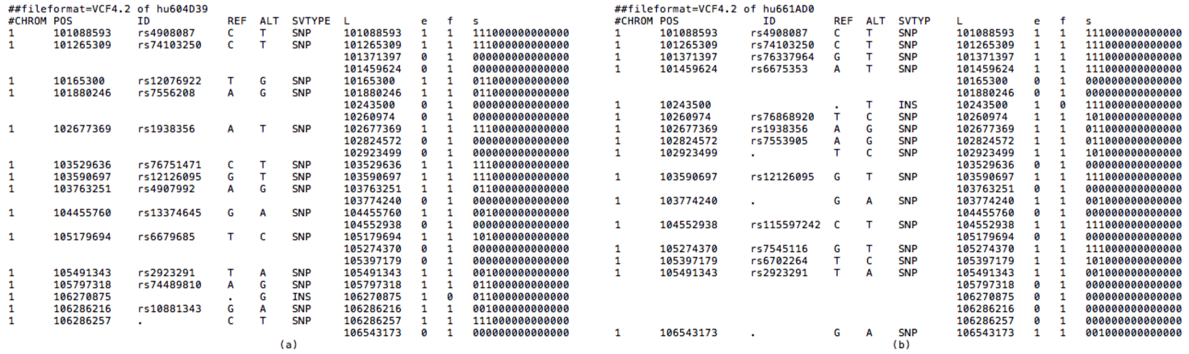
```
#case000 #case001 #case002 #case003 #case004 #case005 #case006 #case007 #case008 #case009 #case010 #case011 #case012 #case013 #case014 #case015 #case016 #case017 #case018
#case019 #case020 #case021 #case022 #case023 #case024 #case025 #case026 #case027 #case028 #case029 #case030 #case031 #case032 #case033 #case034 #case035 #case036 #case037
#case038 #case039 #case040 #case041 #case042 #case043 #case044 #case045 #case046 #case047 #case048 #case049 #case050 #case051 #case052 #case053 #case054 #case055 #case056
#case057 #case058 #case059 #case060 #case061 #case062 #case063 #case064 #case065 #case066 #case067 #case068 #case069 #case070 #case071 #case072 #case073 #case074 #case075
#case076 #case077 #case078 #case079 #case080 #case081 #case082 #case083 #case084 #case085 #case086 #case087 #case088 #case089 #case090 #case091 #case092 #case093 #case094
#case095 #case096 #case097 #case098 #case099 #case100 #case101 #case102 #case103 #case104 #case105 #case106 #case107 #case108 #case109 #case110 #case111 #case112 #case113
#case114 #case115 #case116 #case117 #case118 #case119 #case120 #case121 #case122 #case123 #case124 #case125 #case126 #case127 #case128 #case129 #case130 #case131 #case132
#case133 #case134 #case135 #case136 #case137 #case138 #case139 #case140 #case141 #case142 #case143 #case144 #case145 #case146 #case147 #case148 #case149 #case150 #case151
#case152 #case153 #case154 #case155 #case156 #case157 #case158 #case159 #case160 #case161 #case162 #case163 #case164 #case165 #case166 #case167 #case168 #case169 #case170
#case171 #case172 #case173 #case174 #case175 #case176 #case177 #case178 #case179 #case180 #case181 #case182 #case183 #case184 #case185 #case186 #case187 #case188 #case189
#case190 #case191 #case192 #case193 #case194 #case195 #case196 #case197 #case198 #case199

rs11686243
AG AG AA AG GG AA AG AA GG AG AA AA AA AA AA AA AA GG GG AG AG AA AG GG AA AA GG AG AG AG GG AG AA AA AG AG AG AG AG AG AA AG GG AG AA GG GG GG GG AG AG AG AG AA GG GG GG
AG AA AG GG AG AA GG GG AG AG AG AG AG AA AA AG AG AG AA AG AG AG AG GG AG AG AG GG GG AG AG GG AG AG AG AA AA GG AG AA GG AA AA AG GG AG AG AG AG AG AG AG AG GG GG AA AG
AG AG AG AA AG GG AG GG AA AG GG AG AG AG AA AG AG AG GG AG GG AG GG AG AG AG GG AG AG GG GG AG AG GG AA GG AA AG AG AG AG GG AG AA AG GG GG AG AG AG AG AG GG AG AG AA AG
AA AA AG GG AA AG AG GG AG GG AG AG GG GG AG AG AA AG AG AG GG AG GG GG AG AG GG AG GG

1  1  2  1  0  2  1  2  0  1  2  2  2  2  2  2  2  0  0  1  1  2  1  0  2  2  0  1  1  1  0  1  2  2  1  1  1  1  1  1  2  1  0  1  2  0  0  0  0  1  1  1  1  2  0  0  0
1  2  1  0  1  0  0  0  1  1  1  1  1  2  2  1  1  1  2  1  1  1  1  0  1  1  1  0  0  1  1  0  1  1  1  2  2  0  1  2  0  2  2  1  0  1  1  1  1  1  1  1  1  0  0  2  1
1  1  1  2  1  0  1  0  2  1  0  1  1  1  2  1  1  1  0  1  0  1  0  1  1  1  0  1  1  0  0  1  1  0  2  0  2  1  1  1  1  0  1  2  1  0  0  1  1  1  1  1  0  1  1  2  1
2  2  1  0  2  1  1  0  1  0  1  1  0  0  1  1  2  1  1  1  0  1  0  0  1  1  0  1  0


rs4426491
CC CC CC CT CT CC CT CC CT CT CC CC CC CC CC CC CC CT CT CT CC CC CT CT CC CC CT CC CT CC CT CC CC CC CT CC CC CT CT CC CC CC CT CT CC CC CT CT CT CC CC CC CC CC CT CC CT
CT CC CT CT CT CC CT CT CT CC CT CT CC CC CC CC CT CC CC CC CT CT CT CT CT CC CT CT CT CT CC CT CT CC CC CC CC CC CT CC CT CC CC CC CT CC CT CC CC CT CC CC CT CC CT CC CC
CC CC CT CC CC CT CC CT CC CC CT CC CC CT CC CT CT CC CC CC CT CT CT CC CT CC CT CT CT CT CT CT CT TT CC CC CC CT CT CT CC CT CC CC CC CT CT CT CT CC CC CT TT CT CT CC CC
CC CC CC TT CC CT CT CT CT CT CT CC CT CC CT CC CC CT CT CC CT CC CT CT CC CC TT CT CT
2  2  2  1  1  2  1  2  1  1  2  2  2  2  2  2  2  1  1  1  2  2  1  1  2  2  1  2  1  2  1  2  2  2  1  2  2  1  1  2  2  2  1  1  2  2  1  1  1  2  2  2  2  2  1  2  1
1  2  1  1  1  2  1  1  1  2  1  1  2  2  2  2  1  2  2  2  1  1  1  1  1  2  1  1  1  1  2  1  1  2  2  2  2  2  1  2  1  2  2  2  1  2  1  2  2  1  2  2  1  2  1  2  2
2  2  1  2  2  1  2  1  2  2  1  2  2  1  2  1  1  2  2  2  1  1  1  2  1  2  1  1  1  1  1  1  1  0  2  2  2  1  1  1  2  1  2  2  2  1  1  1  1  2  2  1  0  1  1  2  2
2  2  2  0  2  1  1  1  1  1  1  2  1  2  1  2  2  1  1  2  1  2  1  1  2  2  0  1  1
```

Fig. 1. A snapshot of the dataset for task 1 and its encodings.



##fileformat=VCF4.2 of hu604D39

#CHROM	POS	ID	REF	ALT	SVTYPE	L	e	f	s
1	101088593	rs4908087	C	T	SNP	101088593	1	1	111000000000000
1	101265309	rs74103250	C	T	SNP	101265309	1	1	111000000000000
						101371397	0	1	000000000000000
						101459624	0	1	000000000000000
1	10165300	rs12076922	T	G	SNP	10165300	1	1	011000000000000
1	101880246	rs7556208	A	G	SNP	101880246	1	1	011000000000000
						10243500	0	1	000000000000000
						10260974	0	1	000000000000000
1	102677369	rs1938356	A	T	SNP	102677369	1	1	111000000000000
						102824572	0	1	000000000000000
						102923499	0	1	000000000000000
1	103529636	rs76751471	C	T	SNP	103529636	1	1	111000000000000
1	103590697	rs12126095	G	T	SNP	103590697	1	1	111000000000000
1	103763251	rs4907992	A	G	SNP	103763251	1	1	011000000000000
						103774240	0	1	000000000000000
1	104455760	rs13374645	G	A	SNP	104455760	1	1	001000000000000
						104552938	0	1	000000000000000
1	105179694	rs6679685	T	C	SNP	105179694	1	1	101000000000000
						105274370	0	1	000000000000000
						105397179	0	1	000000000000000
1	105491343	rs2923291	T	A	SNP	105491343	1	1	001000000000000
1	105797318	rs74489810	A	G	SNP	105797318	1	1	011000000000000
1	106270875		.	G	INS	106270875	1	0	011000000000000
1	106286216	rs10881343	G	A	SNP	106286216	1	1	001000000000000
1	106286257	.	C	T	SNP	106286257	1	1	111000000000000
						106543173	0	1	000000000000000

(a)

##fileformat=VCF4.2 of hu661AD0

#CHROM	POS	ID	REF	ALT	SVTYP	L	e	f	s
1	101088593	rs4908087	C	T	SNP	101088593	1	1	111000000000000
1	101265309	rs74103250	C	T	SNP	101265309	1	1	111000000000000
1	101371397	rs76337964	G	T	SNP	101371397	1	1	111000000000000
1	101459624	rs6675353	A	T	SNP	101459624	1	1	111000000000000
						10165300	0	1	000000000000000
						101880246	0	1	000000000000000
1	10243500		.	T	INS	10243500	1	0	111000000000000
1	10260974	rs76868920	T	C	SNP	10260974	1	1	101000000000000
1	102677369	rs1938356	A	G	SNP	102677369	1	1	011000000000000
1	102824572	rs7553905	A	G	SNP	102824572	1	1	011000000000000
1	102923499	.	T	C	SNP	102923499	1	1	101000000000000
						103529636	0	1	000000000000000
1	103590697	rs12126095	G	T	SNP	103590697	1	1	111000000000000
						103763251	0	1	000000000000000
1	103774240	.	G	A	SNP	103774240	1	1	001000000000000
						104455760	0	1	000000000000000
1	104552938	rs115597242	C	T	SNP	104552938	1	1	111000000000000
						105179694	0	1	000000000000000
1	105274370	rs7545116	G	T	SNP	105274370	1	1	111000000000000
1	105397179	rs6702264	T	C	SNP	105397179	1	1	101000000000000
1	105491343	rs2923291	T	A	SNP	105491343	1	1	001000000000000
						105797318	0	1	000000000000000
						106270875	0	1	000000000000000
						106286216	0	1	000000000000000
						106286257	0	1	000000000000000
1	106543173	.	G	A	SNP	106543173	1	1	001000000000000

(b)

Fig. 2. A snapshot of the dataset for task 2 and its encodings: (a)hu604D39 and (b)hu661AD0

The value e_i defines whether the genotype at the specified locus is missing; the value f_i specifies the variants compared with the reference.

Since both VCF files are aligned with the same reference genome, we don't need to compare the columns of 'REF'. To improve performance, we assume that it suffices to compare 7 SNPs between two non-reference sequences. In the following, we describe how to encode the sequences. Each SNP is represented by two bits as

$$A \to 00, \; G \to 01, \; C \to 10, \; T \to 11,$$

and then concatenated with each other. Next we pad with 1 at the end of the bit string so as to distinguish the A-strings. Finally, we pad with zeros to make it a binary string of length 15, denoted by $\mathbf{s}_i$. Let $\mathbf{s}_i[j]$ denote j-th bit of $\mathbf{s}_i$. If a person's SNV at the given locus is not known (*i.e.,* $e_i = 0$), then it is encoded as 0-string. For example, 'GTC' is encoded as a bit string $01||11||10||10\ldots0$, of length 15.

Finally, let us consider the i-th genotype lengths D_i, D'_i of two participants defined as follows: when it has no variants at the given locus of the sequence, set zero as the length at the locus. If it includes a deletion compared with the reference, use the length of reference. Otherwise, we take the length of the target sequence at the current locus. In Figure 2, we illustrate the file format of the data for task 2 and its encodings.

4.2 Homomorphic Computation of The BGV Scheme

We describe how to compute the genomic algorithms described above on encrypted genetic data using the BGV scheme.

4.2.1 Task 1: GWAS on Encrypted Genomic Data

Using the encodings that we propose for practical HE, we can homomorphically evaluate any function involving additions and multiplications, but it is not known how to perform homomorphic division of integer values. We obtain the counts using a few homomorphic additions.

Let g_j be the encoded value of SNV site j based on the encoding method as described in Section 4.1. Then each person packs g_j into the j-th slot. Let s be the total number of SNVs. Assuming that each ciphertext holds ℓ plaintext slots for $s \leq \ell$, the i-th person encrypts the vector $(g_i^{(1)}, \ldots, g_i^{(s)}, 0 \ldots, 0) \in \mathbb{Z}_t^\ell$ using batching as

$$\mathsf{ct}_i = \mathsf{Enc}(\mathsf{CRT}((g_i^{(1)}, \ldots, g_i^{(s)}, 0 \ldots, 0), \mathsf{pk}).$$

Let ct_{eval} be a ciphertext given by the homomorphic operation

$$\mathsf{ct}_{eval} = \sum_{i=1}^{N} \mathsf{ct}_i. \tag{1}$$

Note that the use of batching technique enables to perform N aggregate operations in parallel. Next, let $\mathsf{m} = \mathsf{Dec}(\mathsf{ct}_{eval}, \mathsf{sk})$ denote the decryption of the ciphertext ct_{eval} and decode the s outputs from the output plaintext polynomial as follows: let m_j be the constant coefficient of $\mathsf{m} \bmod f_j$ for $1 \le j \le s$. That is, we have

$$\mathsf{m}_j \overset{let}{=} \mathsf{m} \bmod f_j = \sum_{i=1}^{N} g_i^{(j)}.$$

Thus the MAF of SNV j in the group is computed as

$$\frac{\min\{\mathsf{m}_j, 2N - \mathsf{m}_j\}}{2N}.$$

For the homomorphic evaluation of χ^2 test, each group performs aggregations over ciphertexts as shown in (1). Let ct_{case} and ct_{cont} denote the ciphertexts by the evaluations in the case and control groups, respectively. Then one can compute two ciphertexts by the homomorphic operations

$$\mathsf{ct}^{+} \overset{let}{=} \mathsf{ct}_{case} + \mathsf{ct}_{cont}, \quad \mathsf{ct}^{-} \overset{let}{=} \mathsf{ct}_{case} - \mathsf{ct}_{cont}.$$

The plaintext polynomial from ct^{+} can be decoded as the plaintext slots which have $(n_A^{(j)} + n_A'^{(j)})$ at the j-th slot. In other words, we have

$$\mathsf{Dec}(\mathsf{ct}^{+}, \mathsf{sk}) \bmod f_j = \sum_{i=1}^{N} (g_i^{(j)} + g_i'^{(j)}) = n_A^{(j)} + n_A'^{(j)}.$$

Similarly, the plaintext polynomial from ct^{-} is decoded as the plaintext slots which has the value congruent to $(n_A - n_A')$ in the interval $[0, t) \cap \mathbb{Z}$. Thus, if the output value is larger than $\frac{t}{2}$, then subtract t from it; that is, we have

$$[\mathsf{Dec}(\mathsf{ct}^{-}, \mathsf{sk}) \bmod f_j]_t = \sum_{i=1}^{N} (g_i^{(j)} - g_i'^{(j)}) = n_A^{(j)} - n_A'^{(j)}.$$

Task 2: Secure DNA Sequence Comparison

We represent sequence comparison algorithms as binary circuits and then evaluate them over encrypted data. We use the native plaintext space of binary polynomials ($i.e.$, $R_2 = \mathbb{Z}_2[x]/(\Phi_m(x))$), and denote XOR and AND as $\oplus$ and $\wedge$, respectively. For simplicity, you may consider the plaintext space $\mathbb{Z}_2^{\ell}$ supporting batching operation with ℓ slots.

For the homomorphic evaluation of Hamming distance, the genomic data of two participants, denoted by $(e_i, f_i, \mathbf{s}_i)$ and $(e_i', f_i', \mathbf{s}_i')$, are encrypted bit-wise. For example, the encryptions of e_i's are in the form of

$$\begin{aligned}
&\mathsf{Enc}\left(\mathsf{CRT}(e_1, \ldots, e_\ell), \mathsf{pk}\right),\\
&\mathsf{Enc}\left(\mathsf{CRT}(e_{\ell+1}, \ldots, e_{2\ell}), \mathsf{pk}\right), \ldots,\\
&\mathsf{Enc}\left(\mathsf{CRT}(e_{\lfloor \ell(\mathcal{L})/\ell \rfloor \cdot \ell+1}, \ldots, e_{\ell(\mathcal{L})}, 0, \ldots, 0), \mathsf{pk}\right).
\end{aligned}$$

This allows to compute the same function on ℓ inputs at the price of one computation. Then one can evaluate the following binary circuit over encryption:

$$\Big(\mathtt{E}(\mathbf{s}_i, \mathbf{s}'_i) \wedge \big(e_i \oplus e'_i \oplus 1\big) \oplus 1\Big) \wedge f_i \wedge f'_i$$

where $\mathtt{E}(\mathbf{s}_i, \mathbf{s}'_i) = \wedge_{j=1}^{15} (\mathbf{s}_i[j] \oplus \mathbf{s}'_i[j] \oplus 1)$ has 1 if and only if $\mathbf{s}_i, \mathbf{s}'_i$ are the same. After homomorphic computations, the output can be decrypted with the secret key. The plaintext polynomial has the Hamming distance result of SNV site i at the i-th slot, so we need only aggregate them.

Now, we consider the comparison binary circuit (described in [11]) for the secure computation of the approximate Edit distance. We express an unsigned μ-bit integer x in its binary representation and denote the j-th coordinate of x by $x[j]$ (*i.e.,* $x = \sum_{j=1}^{\mu} x[j] \cdot 2^{j-1}, x[j] \in \{0, 1\}$). For two μ-bit integers x and y, the comparison circuit is defined by

$$\mathtt{C}(x, y) = \begin{cases} 1 & \text{if } x < y, \\ 0 & \text{o.w.}, \end{cases}$$

and this is written recursively as $\mathtt{C}(x, y) := c_\mu$ where

$$c_j = \big((x[j] \oplus 1) \wedge y[j]\big) \oplus \big((x[j] \oplus 1 \oplus y[j]) \wedge c_{j-1}\big)$$

for $j \geq 2$ with an initial value $c_1 = (x[1] \oplus 1) \wedge y[1]$. Then the j-th bit of maximum value between two inputs is defined as follows:

$$\begin{aligned} \mathtt{max}\{x, y\}[j] &= \big((1 \oplus \mathtt{C}(x, y)) \wedge x[j]\big) \oplus \big(\mathtt{C}(x, y) \wedge y[j]\big) \\ &= x[j] \oplus \big(\mathtt{C}(x, y) \wedge (x[j] \oplus y[j])\big). \end{aligned}$$

For the bit-sliced implementation, all the lengths are also expressed in a binary representation and we denote the maximum length of SNPs by μ. It follows from the primitive circuits that we can evaluate the circuits homomorphically:

$$\Big(\mathtt{E}(\mathbf{s}_i, \mathbf{s}'_i) \wedge (f_i \oplus f'_i \oplus 1) \oplus 1\Big) \ \wedge \ \mathtt{max}\{D_i, D'_i\}[j].$$

Finally, one can decrypt the results and decode $\ell(\mathcal{L})$ values from the output plaintext polynomials. More precisely, let $\ell_{i,j}$ be the value at i-th slot which corresponds to the j-th bit. We see that $\sum_{j=1}^{\mu} \ell_{i,j} \cdot 2^{j-1}$ is the approximate Edit distance of SNV site i, hence we need only perform aggregation operations over them.

4.3 Homomorphic Computation of The YASHE Scheme

We explain how to evaluate the genomic algorithms homomorphically using the YASHE scheme.

4.3.1 Task 1: GWAS on Encrypted Genomic Data

Lauter et al. [21] introduced a method how to pack m bits $b_0, \ldots, b_{m-1}$ into a single ciphertext that encodes the polynomial $b(x) = \sum_{i=0}^{m-1} b_i x^i$. We note that polynomial addition corresponds to simple component-wise addition of the vectors. Since a case-control study requires only additions, this method can be used for our case. When using a ring polynomial $x^n + 1$ with a power-of-two n, we can embed data of $n' \overset{let}{=} \lfloor \frac{n}{s} \rfloor$ persons into a single plaintext polynomial. Namely, one can encrypt the polynomial

$$\mathsf{pm}(g_1 = (g_1^{(1)}, \ldots, g_1^{(s)}), \ \ldots \ , g_{n'} = (g_{n'}^{(1)}, \ldots, g_{n'}^{(s)})) \overset{let}{=} \sum_{i=1}^{n'} \sum_{j=0}^{s-1} g_i^{(j)} x^{j + s \cdot (i-1)}.$$

The simple aggregation operations are performed over packed ciphertexts. Now, let

$$\mathsf{m} = \sum_{j=0}^{n's-1} \mathsf{m}_j x^j \in R_t$$

denote the decryption result of the evaluated ciphertext. Then, for $1 \leq j \leq s$, one can aggregate n' data from the output plaintext polynomial by computing

$$\mathsf{m}_j \leftarrow \sum_{i=0}^{n'-1} \mathsf{m}_{j+is},$$

which is the allele counts of A at the SNV site j. Notice that if $n' = 1$, then we don't need to do the above operations. Hence, the MAF of the SNV j in the group is computed as

$$\frac{\min\{\mathsf{m}_j, 2N - \mathsf{m}_j\}}{2N}.$$

Similarly, let ct^+ and ct^- denote the ciphertexts computed by the homomorphic additions and subtractions after simple aggregations. As we have demonstrated, we need additional aggregation processes after decryptions. Let

$$\mathsf{m}^+ = \sum_{j=0}^{n's-1} \mathsf{m}_j^+ x^j, \quad \mathsf{m}^- = \sum_{j=0}^{n's-1} \mathsf{m}_j^- x^j$$

denote the decryption polynomials of ct^+ and ct^-, respectively. Then, for $1 \leq j \leq s$, one can obtain the allele counts by computing as

$$n_A^{(j)} + n_A'^{(j)} = \sum_{i=0}^{n'-1} \mathsf{m}_{j+is}^+, \quad n_A^{(j)} - n_A'^{(j)} = \left[\sum_{i=0}^{n'-1} \mathsf{m}_{j+is}^- \right]_t.$$

4.3.2 Task 2: Secure DNA Sequence Comparison

Since polynomial multiplication does not correspond to component-wise multiplication of the vectors, we have to consider another packing method instead of [21]. Let us consider the polynomial-CRT packing method. The m-th cyclotomic polynomial $\Phi_m(x)$ factors modulo 2 into a product of the same irreducible factors (*i.e.,* $\Phi_m(x) = x^n + 1 = (x+1)^n \bmod 2$), so we cannot apply batching technique with these parameters. We can instead do that if taking a prime t (not 2) such that the polynomial splits into the distinct factors modulo t, but the use of a different message space leads to change our primitive circuits.

As noted in [10], we see that for $x, y \in \{0,1\}$, the following properties hold: $x \oplus y = (x-y)^2$ and $x \wedge y = x \cdot y$ where $-$ and $\cdot$ are arithmetic operations over integers. From these observations, we can amend the evaluation circuit for the Hamming distance as follows:

$$\left(\mathtt{E}(\mathbf{s}_i, \mathbf{s}_i') \cdot \left((e_i - e_i')^2 - 1 \right) + 1 \right) \cdot f_i \cdot f_i'$$

where $\mathtt{E}(\mathbf{s}_i, \mathbf{s}_i') = \prod_{j=1}^{15} \left(1 - (\mathbf{s}_i[j] - \mathbf{s}_i'[j])^2\right)$.

We note that for μ-bit integer x and y, the comparison circuit $\mathtt{C}(x, y) = c_\mu$ can be expressed as

$$c_j = (1 - x[j]) \cdot y[j] + \left(1 - (x[j] - y[j])^2\right) \cdot c_{j-1}.$$

for $j \geq 2$ with $c_1 = (1 - x[1]) \cdot y[1]$. Since it is available to compute on large integer inputs, the maximum value is defined by

$$\begin{aligned}\mathtt{max}\{x, y\} &= (1 - \mathtt{C}(x, y)) \cdot x + \mathtt{C}(x, y) \cdot y \\ &= x + \mathtt{C}(x, y) \cdot (y - x).\end{aligned}$$

Using these circuits, we compute the ciphertext given by the homomorphic operations

$$\Big(1 + \mathtt{E}(\mathbf{s}_i, \mathbf{s}'_i) \cdot \big((f_i - f'_i)^2 - 1\big)\Big) \cdot \mathtt{max}\{D_i, D'_i\}.$$

Then we get the encryptions of the approximate Edit distance result of SNV i.

5 Experimental Results & Discussion

In this section, we explain how to set the parameters for homomorphic evaluations and present our experimental results. We used BGV scheme with Shoup-Halevi's HE library [17] (called HELib). HELib is written in C++ and based on the arithmetic library NTL [28] over GMP. Our experiments with BGV were performed on a Linux machine with an Intel Xeon 2.67 GHz processor. We also implemented YASHE scheme with ARITH library in C. The measurements were done in an Intel Core 3.60GHz, running 64-bit Windows 7.

The dataset used for task 1 consists of 200 case group (constructed from 200 participants from PGP) and 200 control group (simulated based on the haplotypes of 174 participants from CEU population of HapMap Project). The dataset for task 2 consists of two individual genomes randomly selected from PGP.

5.1 Theoretical Comparison between BGV and YASHE

BGV scheme has a chain of ciphertext moduli by a set of primes of roughly the same size, $p_0, \cdots, p_{L-1}$, that is, the i-th modulus q_i is defined as $q_i = \prod_{k=0}^{i} p_k$. For simplicity, assume that p is the approximate size of the p_is. Given the lattice dimension $n = \phi(m)$, the plaintext modulus t, and the Hamming weight h of the secret key, it follows from Theorem 3 in [10] that the depth of a classical homomorphic multiplication is

$$d_{n,t} \approx \left\lceil \frac{\log_2(h \cdot n \cdot t^4)}{2\log_2(p)} \right\rceil \approx \left\lceil \frac{\log_2(h \cdot n \cdot t^4)}{36} \right\rceil,$$

so the total number of modulus switching operations during the M-levels of multiplications is about $\mathsf{M} \cdot d_{n,t}$. Since we first should do one modulus switching to the initial ciphertext before homomorphic computation, we see that $L = \mathsf{M} \cdot d_{n,t} + 2$. Thus we can approximate the size of the ciphertext modulus q_{BGV} in the BGV scheme (from C.3 in [14]) as follows:

$$\log_2 q_{\mathsf{BGV}} \approx 24 + \frac{3}{2}\log_2 n + (L-2) \cdot (11 + \frac{1}{2}\log_2 n) < (L+1) \cdot (11 + \frac{1}{2}\log_2 n).$$

Since a fresh ciphertext in BGV consists of a pair of polynomials over $R_{q_{L-1}}$, the size of ciphertext from the above inequality is about

$$|\mathsf{ct}|_{\mathsf{BGV}} \approx 2n \cdot \log_2 q_{\mathsf{BGV}} \approx 2n(L+1) \cdot (11 + \frac{1}{2}\log_2 n).$$

Similarly, [6, Lemma 9] provides a theoretical upper bound on the noise growth after M multiplicative levels for YASHE as $(nt)^{2(\mathsf{M}-1)} \cdot (12n^2 t\sigma\ell_{\omega,q}\omega\mathsf{M})$ when taking $B = 6\sigma$ as the

Table 2. The theoretical sizes of ciphertext modulus and a ciphertext

	BGV	YASHE
$\log_2 q$	$(\mathsf{M} \cdot \frac{\log_2(h \cdot n \cdot t^4)}{36} + 3) \cdot (11 + \frac{1}{2} \log_2 n)$	$2\mathsf{M} \cdot \log_2 nt$
$\lvert\mathsf{ct}\rvert$	$2n(\mathsf{M} \cdot \frac{\log_2(h \cdot n \cdot t^4)}{36} + 3) \cdot (11 + \frac{1}{2} \log_2 n)$	$2n\mathsf{M} \cdot \log_2 nt$

coefficient bound of error polynomials. It should be less than the ratio of q_{YASHE} to t so that the decryption procedure works; we should select a ciphertext modulus q_{YASHE} so as to satisfy

$$\log_2 q_{\mathsf{YASHE}} \approx 2\mathsf{M} \cdot \log_2 nt + \log_2(12\sigma \ell_{\omega,q} \omega \mathsf{M}) \geq 2\mathsf{M} \cdot (\log_2 nt).$$

Since a ciphertext consists of only a single ring element, the size is about

$$|\mathsf{ct}|_{\mathsf{YASHE}} \approx n \cdot \log_2 q_{\mathsf{YASHE}} \approx 2n\mathsf{M} \cdot (\log_2 nt).$$

We summarize the above results in Table 2.

Note that it is difficult to compare these two schemes because their parameters depend on at least 4 variables: the plaintext modulus, t, the dimension, n, the Hamming weight, h, and the number of multaplicative levels to be evaluated, M. However we observe that, in the case that $\log_2 n \approx 14$ and $h = 64$, we have:

$$\begin{aligned}
\log_2 q_{\mathsf{YASHE}} - \log_2 q_{\mathsf{BGV}} &\approx 2\mathsf{M} \cdot (\log_2 nt) - (\mathsf{M} \cdot d_{n,t} + 3) \cdot (11 + \frac{1}{2} \log_2 n) \\
&\approx 2\mathsf{M} \cdot (14 + \log_2 t) - (\mathsf{M} \cdot d_{n,t} + 3) \cdot 18 \\
&= 2\mathsf{M} \cdot (14 + \log_2 t - 9 \cdot d_{n,t}) - 54 \\
&\approx 2\mathsf{M} \left(14 + \log_2 t - 9 \cdot \left(\frac{20 + 4\log_2 t}{36} + \eta\right)\right) - 54 \\
&= 18\mathsf{M}(1 - \eta) - 54 \quad \text{for some} \;\; 0 \leq \eta < 1.
\end{aligned}$$

Hence, if M is large, we can use a smaller ciphertext modulus to evaluate M-levels of multiplications with BGV in comparison to YASHE; however, the YASHE scheme has smaller ciphertexts than BGV. This follows from the fact that

$$\begin{aligned}
|\mathsf{ct}|_{\mathsf{BGV}} - |\mathsf{ct}|_{\mathsf{YASHE}} &\approx 2(\mathsf{M} \cdot d_{n,t} + 3) \cdot (11 + \frac{1}{2} \log_2 n) - 2\mathsf{M} \cdot (\log_2 nt) \\
&\approx 2\mathsf{M} \cdot (18 \cdot d_{n,t} - 14 - \log_2 t) + 108 \\
&\approx 2\mathsf{M} \cdot (\log_2 t + 18\eta - 4) + 108
\end{aligned}$$

for some $0 \leq \eta < 1$; if $\log_2 t \geq 4$, then $\log_2 t + 18\eta - 4 \geq 0$; otherwise, we have $d_{n,t} = 1$ and so $18 \cdot d_{n,t} - 14 - \log_2 t > 0$.

Let us contrast the complexity of homomorphic multiplication operations for the two schemes. One of the new optimizations for BGV is to convert polynomials between coefficient and evaluation representations. Most of the homomorphic operations are performed in the more efficient evaluation representation, but it sometimes requires coefficient representation. Note that these conversions take the most time in execution. In more detail, at the l-th level of this scheme, the key switching procedure requires $\mathcal{O}(l)$ Fast Fourier Transforms (FFTs) and the modulus switching operation requires $(l + 1)$ FFTs. Since HElib uses the Bluestein FFT algorithm [5] (with run-time complexity of $\mathcal{O}(n \log n)$), this yields an overall complexity of $\mathcal{O}(ln \log n)$ for a multiplication of ciphertexts.

For the polynomial multiplication in the base ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$, we implemented the FFT algorithm by Nussbaumer [27] based on recursive negacyclic convolutions (with run-time complexity $\frac{9}{2}n \log n \log\log n\ + \mathcal{O}(n \log n)$ of arithmetic operations in $\mathbb{Z}_q$). The homomorphic multiplication in YASHE includes a costly key switching operation which is an inner product on $R_q^{\ell_{\omega,q}}$, hence we obtain a total cost of $\ell_{\omega,q} \cdot (\frac{9}{2}n \log n \log\log n + \mathcal{O}(n \log n))$ operations for a ciphertext multiplication. Therefore, BGV is expected to be faster than YASHE for a ciphertext multiplication if we take similar parameters with q and n.

5.2 How to Set Parameters

The security of BGV relies on the hardness of the RLWE assumption. Similarly, YASHE is provably secure in the sense of IND-CPA under the RLWE assumption and DSPR assumption. The main difference between the schemes is that BGV uses an odd integer m while YASHE chooses m to be a power-of-two with a prime integer q such that $q \equiv 1 \pmod{m}$. In [25], it was shown that the hardness of RLWE with the cyclotomic polynomial $\Phi_m(x) = x^{\phi(m)} + 1$ can be established by a quantum reduction to shortest vector problems in ideal lattices. This means that YASHE is believed to be secure as long as the lattice problems are hard to solve.

5.2.1 Parameters of the BGV scheme

To homomorphically evaluate the algorithms for task 1, we first choose sufficiently large t so that no reductions modulo t occurs in the plaintext slots. For example, we take t as the smallest power-of-two which satisfies the following inequalities:

$$n_A^{(j)} = \sum_{i=1}^{200} g_i^{(j)} \le \sum_{i=1}^{200} 2 = 400 < t$$

since the total number of people in the same group is $N = 200$. So it suffices to take $t = 2^9$ for privately computing the minor allele counts. In the case of χ^2 test, we have

$$n_A^{(j)} + n_A'^{(j)} = \sum_{i=1}^{200} g_i^{(j)} + \sum_{i=1}^{200} g_i'^{(j)} \le 2\sum_{i=1}^{200} 2 = 800 < t,$$

thus we set the parameter $t = 2^{10}$. For the second task, we used $t = 2$ to evaluate binary circuits.

Now, we derive a lower-bound on $\phi(m)$ such that

$$\phi(m) \ge \frac{(L(\log m + 23) - 8.5) \cdot (\lambda + 110)}{7.2}. \tag{2}$$

from the security analysis of [14] based on Lindner and Peikert's method [23]. For the efficiency of the implementation, we choose the smallest integer m so as to satisfy Inequality (2) and pack the message into plaintext slots as many as possible. Next, we define a ladder of moduli to make the correct decryption after computation with L levels (see [14] for details). Finally, we consider the discrete Gaussian distribution $\chi_{err} = D_{\mathbb{Z},\sigma}$ with mean 0 and standard deviation $\sigma = 3.2$ over the integers to sample random error polynomials.

5.2.2 Parameters of the YASHE Scheme

As discussed before, $t = 2^{10}$ will suffice to compute the MAFs and χ^2 statistic. For the second task, we look for the parameter $t \neq 2$ which maximizes the number of slots we can handle in

Table 3. Implementation results of task 1 using BGV and YASHE

		s	t	$\log_2 q$	n	ℓ	L	|ct|	KeyGen	Encrypt	**Eval**	Decrypt
BGV	**MAF**	311	2^9	60	5292	378	3	78kB	6.92s	11.90s	**29.99ms**	290.06ms
		610		61	8190	630		122kB	10.28s	14.85s	**33.36ms**	690.23ms
	χ^2	311	2^{10}	60	5292	378	3	78kB	6.35s	11.61s	**30.05ms**	560.10ms
		610		61	8190	630		122kB	12.27s	15.13s	**38.17ms**	720.33ms
YASHE	**MAF**	311	2^{10}	48	1024	1024	0	6kB	0.01s	1.63s	**5.74ms**	33.71ms
		610							0.04s	4.10s	**16.98ms**	16.78ms
	χ^2	311							0.01s	1.61s	**5.99ms**	16.73ms
		610							0.04s	4.12s	**17.20ms**	17.01ms

one go. We fix the word $\omega = 2^{128}$ for the evaluation key and the standard deviation $\sigma = 8$ for the error distribution χ_{err}.

Since we can estimate the size of noise during homomorphic operations, we get the lower bound on q to ensure the correctness. We also have maximal values of q to ensure the desired security using the results of [22], so that we can have more loose bound than that from LP's method. Then we set m as a power-of-two to get a non-trivial interval for q and then select a smallest q in this interval.

5.3 Implementation Results

We present the parameter setting and performance results for secure genome analysis in Table 3 and 4. All the parameters provide 80-bit security level. We give the plaintext modulus t, the size of the ciphertext modulus q, the lattice dimension $n = \phi(m)$, and the number of plaintext slots ℓ. We also give the circuit depth L so that HE scheme can correctly evaluate such a computation on encrypted data. In particular, it can be considered as the number of ciphertext moduli in the BGV scheme. We consider the ciphertext size in kBytes for a set of parameters. The last columns give the timings for the key generation, encryption, evaluation and decryption.

5.3.1 Performance results of task 1

In Table 3, the top four rows refer to the results using BGV, and the bottom four rows refer to results using YASHE for computing the MAFs and χ^2 statistic in case-control groups. Note that the number of slots means that how many messages we can pack into one single ciphertext. When using YASHE, we can evaluate simultaneously by embedding the data into the coefficients of plaintext polynomial; the maximal degree of plaintext polynomial in this case is considered to be the number of slots.

In practice, we need to apply one more modulus-switching during homomorphic additions for the BGV scheme, so the total number of ciphertext moduli is $L = 1+2 = 3$. On the contrary, L means the levels of multiplications in YASHE (without taking into account the additions). In other words, when evaluating a polynomial of degree d on encrypted data, we have $L \approx \log d$ levels of multiplications by computing in a binary tree way. Thus, $L = 0$ suffices to support such homomorphic additions in task 1. Thus we don't need to generate the evaluation key, which enables to take less time for key generation than BGV. Moreover, the evaluation performance of YASHE is much better since BGV requires a costly modulus switching operations even for computing simple homomorphic additions.

Table 4. Implementation results of task 2 using BGV and YASHE

		Size	t	$\log_2 q$	n	ℓ	L	|ct|	KeyGen	Encrypt	**Eval**	Decrypt
BGV	**Hamming**	5K	2	132	8190	630	7	264kB	2.53s	12.65s	**15.39s**	0.64s
		10K								24.90s	**29.39s**	1.29s
	Edit	5K		150			8	300kB	3.41s	16.98s	**40.86s**	2.97s
		10K								33.34s	**76.08s**	5.81s
YASHE	**Hamming**	5K	8191	384	8192	4096	6	384kB	130.59s	29.70s	**68.31s**	2.67s
		10K								58.82s	**134.87s**	5.04s
	Edit	5K								58.46s	**110.18s**	2.66s
		10K								116.61s	**245.04s**	5.07s

5.3.2 Performance results of task 2

Table 4 presents the parameter setting and performance results for secure DNA sequence comparison using BGV and YASHE. We evaluated the performance with the input data of different sizes $5K$ and $10K$. We implemented the comparison circuit with the same method as described in [11, Lemma 1] in order to reduce the circuit depth over encryption.

As discussed before, given the parameter L, we obtain the approximate size of ciphertext modulus as $\log_2 q \approx 43 + 18 \cdot (L-2)$ for BGV when using $t = 2$ and $R = \mathbb{Z}[x]/(\Phi_{8191}(x))$. Since it should support $L = 7$ or 8 to correctly evaluate genomic algorithms of task 2, we use the modulus q around 130 to 150. On the other hand, the size of the parameter q in YASHE should be strictly larger than $2L\log_2(nt) \approx 52L$ with $t = 2^9$ and $R = \mathbb{Z}[x]/(x^{8192}+1)$. So we used a 384-bit prime q such that $q \equiv 1 \pmod{2^{14}}$.

In the implementation of YASHE scheme, computing the inverse of f modulo q turns out to be the most-time consuming part of the key-generation, which runs in around 128.34 seconds(s). In total, it takes about 130.59s to generate the public key, secret key and evaluation keys, while the key generation of the BGV scheme takes about 3.41s in order to support 8 levels.

There is also quite a big gap between the two schemes in timings for a multiplication of ciphertexts: BGV takes around 0.07s, while YASHE takes around 1.75s (including the key switching step) under the parameter settings used in task 2. For the efficiency of the YASHE scheme, we might avoid a costly key switching step during the homomorphic multiplication; however, it supports a limited number of homomorphic multiplications without the key switching step. This follows since the noise grows exponentially with the multiplicative depth through such consecutive operations. One alternative is to use a *hybrid approach*, in which we leave out key switching in certain places but do it in others using the evaluation key with a power of the secret key so that one can keep the ciphertext noise small for correct decryption. As a result, polynomial multiplication modulo $x^n + 1$ takes about 0.64s, but it is still slower than that in BGV. As expected, BGV is faster than YASHE to evaluate the genomic algorithms for DNA sequence comparison.

6 Conclusion

In this paper, we discussed how to privately perform genomic tests on encrypted genome data using homomorphic encryption. In addition to the efficient implementations of BGV and YASHE, we compared two schemes both theoretically and practically. We found that there is a trade-off between the security and performance. YASHE uses a power-of-two dimension n which defines the $2n$-th cyclotomic polynomial; this is a good choice for providing strong security, but it requires larger parameters to ensure correctness than BGV, and the homomorphic multiplication in

YASHE is slower than that in BGV. Therefore, the performance numbers for BGV are better than YASHE when homomorphically evaluating deep circuits (like the Hamming distance algorithm or approximate Edit distance algorithm). On the other hand, it might be more efficient to use the YASHE scheme for a low-degree computation, such as minor allele frequencies or χ^2 test statistic in a case-control study.

Acknowledgements. The authors would like to thank Michael Naehrig for extensive assistance with the code for the YASHE-based implementation for the contest. 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.

References

1. Hapmap project. http:// hapmap.ncbi.nlm.nih.gov/.
2. Personal genome project. http://www. personalgenomes.org/.
3. E. Ayday, J. L. Raisaro, P. J. McLaren, J. Fellay, and J.-P. Hubaux. Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data. In *2013 USENIX Workshop on Health Information Technologies*.
4. P. Baldi, R. Baronio, and E. D. Cristofaro. Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM conference on Computer and communications security 2011*, pages 691–702.
5. L. I. Bluestein. *A linear filtering approach to the computation of the discrete fourier transform*, volume 18.
6. J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Proceedings of Cryptography and Coding - 14th IMA International Conference 2013*, volume 8308, pages 45–64.
7. J. W. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014.
8. Z. Brakerski. Fully homomorphic encryption without modulus swithing from classical gapsvp. In *Proceedings of Advances in Cryptology-Crypto 2012*, volume 7417, pages 868–886.
9. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference 2012*, pages 309–325.
10. J. H. Cheon, M. Kim, and M. Kim. Search-and-compute on encrypted data. In M. Brenner, N. Christin, B. Johnson, and K. Rohloff, editors, *Proceedings of Financial Cryptography and Data Security - FC 2015 International Workshop WAHC*, volume 8976, pages 142–159.
11. J. H. Cheon, M. Kim, and K. Lauter. Homomorphic computation of edit distance. In M. Brenner, N. Christin, B. Johnson, and K. Rohloff, editors, *Proceedings of Financial Cryptography and Data Security - FC 2015 International Workshop WAHC*, volume 8976, pages 194–212.
12. Y. Erlich and A. Narayanan. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics 2014*, 15:409–421.
13. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 40th ACM symposium on Theory of computing 2009*, pages 169–178.
14. C. Gentry, S. Halevi, and N. Smart. Homomorphic evaluation of the AES circuit. In *Proceedings of Advances in Cryptology-Crypto 2012*, volume 7417, pages 850–867.
15. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Proceedings of Advances in Cryptology-Crypto 2013*, volume 8042, pages 75–92.
16. T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *Proceedings of Information Security and Cryptology-ICISC 2012*, volume 7839, pages 1–21.
17. S. Halevi and V. Shoup. Design and implementation of a homomorphic encryption library. Technical report, IBM, 2013.
18. M. Humbert, E. Ayday, J.-P. Hubaux, and A. Telenti. Addressing the concerns of the lacks family: quantification of kin genomic privacy. In *Proceedings of the ACM SIGSAC conference on Computer and communications security 2013*, pages 1141–1152.
19. M. Kantarcioglu, W. Jiang, Y. Liu, and Malin.B. A cryptographic approach to securely share and query genomic sequences. *IEEE Trans on Inf Technol Biomed 2008*, 12(5):606–617, 2008.

20. K. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. In *Proceedings of Progress in Cryptology - LATINCRYPT 2014*, volume 8895, pages 3–27.
21. K. Lauter, M. Naehrig, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 18th ACM conference on Cloud computing security 2011*, pages 113–124.
22. T. Lepoint and M. Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In *Proceedings of Progress in Cryptology- AFRICACRYPT 2014*, volume 8469, pages 318–335.
23. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Proceedings of Topics in Cryptology- CT-RSA 2011*, volume 6558, pages 319–339.
24. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 40th ACM symposium on Theory of computing 2012*, pages 1219–1234.
25. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Proceedings of Advances in Cryptology-Eurocrypt 2010*, volume 6110, pages 1–23.
26. M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, and X. Wang. Privacy in the genomic era.
27. H. J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Trans on Acoustics, Speech and Signal Proceesing*, 28(2):205–215, 1980.
28. V. Shoup. NTL: A library for doing number theory. http://www.shoup.net/ntl, 2009.
29. N. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *IACR Cryptology ePrint Archive*, 2011(133).
30. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of Advances in Cryptology-Eurocrypt 2010*, volume 6110, pages 24–43.
31. R. A. Wagner and M. J. Fischer. The string to string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.
32. M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM Cloud Computing Security Workshop*, pages 65–76.