BMC Medical Genomics

RESEARCH Open Access

Privacy-preserving approximate GWAS computation based on homomorphic encryption

Duhyeong Kim, Yongha Son, Dongwoo Kim, Andrey Kim, Seungwan Hong and Jung Hee Cheon*

From 7th iDASH Privacy and Security Workshop 2018
San Diego, CA, USA. 15 October 2018

Abstract

Background: One of three tasks in a secure genome analysis competition called iDASH 2018 was to develop a solution for privacy-preserving GWAS computation based on homomorphic encryption. The scenario is that a data holder encrypts a number of individual records, each of which consists of several phenotype and genotype data, and provide the encrypted data to an untrusted server. Then, the server performs a GWAS algorithm based on homomorphic encryption without the decryption key and outputs the result in encrypted state so that there is no information leakage on the sensitive data to the server.

Methods: We develop a privacy-preserving semi-parallel GWAS algorithm by applying an approximate homomorphic encryption scheme HEAAN. Fisher scoring and semi-parallel GWAS algorithms are modified to be efficiently computed over homomorphically encrypted data with several optimization methodologies; substitute matrix inversion by an adjoint matrix, avoid computing a superfluous matrix of super-large size, and transform the algorithm into an approximate version.

Results: Our modified semi-parallel GWAS algorithm based on homomorphic encryption which achieves 128-bit security takes 30–40 minutes for 245 samples containing 10,000–15,000 SNPs. Compared to the true *p*-value from the original semi-parallel GWAS algorithm, the F_1 score of our *p*-value result is over 0.99.

Conclusions: Privacy-preserving semi-parallel GWAS computation can be efficiently done based on homomorphic encryption with sufficiently high accuracy compared to the semi-parallel GWAS computation in unencrypted state.

Keywords: Homomorphic encryption, Privacy, GWAS, Fisher scoring

Background

After the successful completion of the Human Genome Project in the early 21st century, high throughput technology on genetic variations has been rapidly developed and widely studied. In particular, through the development of microarray chip with rather small computational cost, it became possible to determine the genotype of millions of single nucleotide polymorphism (SNP), a variation in a single nucleotide that occurs at a specific position in the genome, for each individual. With those statistical data of genotypes, many researches are proposed that investigate associations between SNPs and phenotypes like major human disease, and especially Genome-wide association study (GWAS) aims to find top significant SNPs relevant to a certain phenotype.

*Correspondence: jhcheon@snu.ac.kr
Department of Mathematical Sciences, Seoul National University, 1, Gwanak-ro, Gwanak-gu, Seoul, Republic of Korea

Motivation

Since genome analysis uses genomic data that are very sensitive and irreplacable, privacy on genomic data has come up to be one of the most important issues in genome analysis including GWAS. The usual privacy-preserving methodology in data analysis is anonymization, perturbation, randomization, and condensation [1]; however, those methods leverage the quality of data with privacy resulting in an inaccurate analysis to some extent. The dilemma regarding the balance between personal privacy and analytical efficiency has been resolved by applying many cryptographic primitives, while homomorphic encryption (HE) is noticed as one of the ultimate cryptographic solutions for privacy-preserving data analysis. Conceptually, HE is an encryption scheme which allows computations over encrypted data without decryption. HE not only fundamentally prevents the leakage of input data during the analysis phase, but also provides an accurate result of analysis since it preserves the original data intactly. However, HE causes a significant blowup of computational cost for analysis, and optimization and modification of algorithm for efficient computation in HE is the main problem of applying HE in data analysis.

Since 2014, there has been an annual biomedical privacy competition hosted by Integrating Data for Analysis, Anonymization and SHaring (iDASH), a national center for biomedical computing in the United States. One of three tasks in iDASH 2018 [2] was to develop a solution for privacy-preserving GWAS computation based on HE, and we participated in this competition with our delicately constructed algorithms.

Summary of results

In this study, we propose approximate HE algorithms for privacy-preserving GWAS computation. To be precise, we transform well-known Fisher scoring and semi-parallel GWAS algorithm into *HE-friendly* algorithms so that we can efficiently evaluate them in encrypted state. Note that the HE-friendly modified Fisher Scoring algorithm can be generally used for logistic regression, not only for GWAS.

The main challenges in transforming the semi-parallel GWAS algorithm (Algorithm 1) to an HE algorithm are complex matrix operations such as multiplication and inversion. Since matrix inversion in HE is complicated and costly, we substitute it by computation of the adjoint matrices and determinant. With this approach, the original Fisher scoring algorithm can be successfully modified to compute encrypted data efficiently. For the efficient computation of semi-parallel GWAS computation based on HE, moreover, we reduced the number of matrix multiplications as many as possible, and modified the original algorithm into an approximate version which requires much less computational cost. The details of our optimization methodologies are well described in "Our optimization methodology" section.

We exploited an approximate HE scheme HEAAN [3, 4] with a publicly available library [5] for the implementation of our modified semi-parallel GWAS algorithm based on HE. The HE algorithm takes about 40 minutes for 245 samples each containing a binary phenotype, 3 covariates, and 14,841 SNPs on Linux with a 2.10GHz processor using 8 threads (4 cores).

Related works

Some works on HE-based genome analyses has been studied over the past years: Lauter et al. [6] studied application of HE on basic genomic algorithms such as the Pearson Goodness-of-Fit test, and Wang et al. [7] performed an exact logistic regression on small datasets based on HE. More recently, some solutions [8–11] submitted to task 3 of iDASH 2017 [12] competition dealt with training the logistic regression model of genomic data based on HE. Some works [13, 14] studied the privacy-preserving GWAS based on HE, but they performed rather simple χ^2 test on quite small numbers of SNPs, which is quite different from iDASH 2018 task; logistic regression on large numbers of SNPs.

There are some other alternative tools to deal with privacy issues in real-world applications. One of them is a cryptographic tool called secure Multi-Party Computation (MPC). In MPC, computations are done online by multiple parties without revealing any information of the result to each party. There have been several remarkable works on privacy-preserving genome analysis based on MPC. In 2017, Jagadeesh et al. [15] proposed privacy-preserving solutions for several genomic diagnose methods based on MPC with practical implementation over real patient data. Recently, Cho, Wu and Berger [16] successfully constructed a practical MPC-based protocol for privacy-preserving GWAS computation over large-scale genomic data. There have been several previous works [14, 17–19] on privacy-preserving GWAS computation based on MPC; however, they had some limitations to be applied in practice since they either require infeasible computational cost or greatly streamlined the task.

Meanwhile, in 2016, Chen et al. [20] proposed a hardware-based solution for privacy-preserving genome analysis with Software Guard Extensions (SGX) [21], the security-related instruction built in Intel CPU which allows secure computations in a private region which cannot be accessed without the private key. They implemented a SGX-based framework for secure transmission disequilibrium test on Kawasaki disease patients with high scalability on the number of SNPs.

Methods

Approximate homomorphic encryption scheme HEAAN

For privacy-preserving GWAS computation, we applied an HE scheme called HEAAN proposed by Cheon et al. [3, 4], which supports *approximate computation* of real numbers in encrypted state. Efficiency of HEAAN in the real world has been proved by showing its application in various fields including machine learning [8, 22, 23] and cyber physical system [24]. The winning solution of iDASH competition in 2017 also applied HEAAN as an HE scheme for privacy-preserving logistic regression on genomic data.

At high-level, for a HEAAN ciphertext ct of some message polynomial $\mathfrak{m}$, the decryption process with a secret key sk is done as

$$\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct}) = \mathfrak{m} + e \approx \mathfrak{m}$$

where e is a small error attached to the message polynomial $\mathfrak{m}$. Furthermore, for ciphertexts ct_1 and ct_2 of message polynomials $\mathfrak{m}_1$ and $\mathfrak{m}_2$, the homomorphic evaluation algorithms $\mathtt{C.Add}$ and $\mathtt{C.Mult}$ satisfy

$$\mathtt{Dec}_{\mathsf{sk}}\left(\mathtt{C.Add}(\mathsf{ct}_1, \mathsf{ct}_2)\right) \approx \mathfrak{m}_1 + \mathfrak{m}_2,$$
$$\mathtt{Dec}_{\mathsf{sk}}(\mathtt{C.Mult}_{\mathsf{evk}}\left(\mathsf{ct}_1, \mathsf{ct}_2)\right) \approx \mathfrak{m}_1 \cdot \mathfrak{m}_2,$$

i.e., addition and multiplication can be *internally* done even in encrypted state.

For formal definitions, let L be a level parameter, and $q_\ell := 2^\ell$ for $1 \le \ell \le L$. Let $R := \mathbb{Z}[X] / \left(X^N + 1\right)$ for a power-of-two N and R_q be a modulo-q quotient ring of R, i.e., $R_q := R/qR$. The distribution $\chi_{\mathtt{key}} := \mathsf{HW}(\mathsf{h})$ over R_q outputs a polynomial of $\{-1, 0, 1\}$-coefficient having h number of non-zero coefficients, and $\chi_{\mathtt{enc}}$ and $\chi_{\mathtt{err}}$ denote the discrete Gaussian distribution with some prefixed standard deviation. We denote the rounding function by $\lfloor \cdot \rceil$, which outputs the nearest integer of a real-number input. For $a = \sum_{i=0}^{N-1} a_i X^i \in \mathbb{R}[X] / \left(X^N + 1\right)$, then $\lfloor a \rceil := \sum_{i=0}^{N-1} \lfloor a_i \rceil X^i \in R$. Finally, $[\cdot]_q$ denotes a component-wise modulo q operation on each element of R_q. Note that those parameters N, L and h satisfying a certain security level can be determined by Albrecht's security estimator [25, 26]. The scheme description of HEAAN is as following:

- $\underline{\mathtt{KeyGen}(\mathsf{params})}$.
  - Sample $s \leftarrow \chi_{\mathtt{key}}$. Set the secret key as $\mathsf{sk} \leftarrow (1, s)$.
  - Sample $a \leftarrow U\left(R_{q_L}\right)$ and $e \leftarrow \chi_{\mathtt{err}}$. Set the public key as $\mathsf{pk} \leftarrow (b, a) \in R_{q_L}^2$ where $b \leftarrow [-a \cdot s + e]_{q_L}$.
  - Sample $a' \leftarrow U\left(R_{q_L^2}\right)$ and $e' \leftarrow \chi_{\mathtt{err}}$. Set the evaluation key as $\mathsf{evk} \leftarrow \left(b', a'\right) \in R_{q_L^2}^2$ where $b' \leftarrow \left[-a's + e' + q_L \cdot s^2\right]_{q_L^2}$.
- $\underline{\mathtt{Enc}_{\mathsf{pk}}(\mathfrak{m})}$. For a message $\mathfrak{m} \in R$, sample $\nu \leftarrow \chi_{\mathtt{enc}}$ and $e_0, e_1 \leftarrow \chi_{\mathtt{err}}$. Output the ciphertext $\mathsf{ct} = [\nu \cdot \mathsf{pk} + (\mathfrak{m} + e_0, e_1)]_{q_L}$.
- $\underline{\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct})}$. For a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{q_\ell}^2$, output a message $\mathfrak{m}' = [c_0 + c_1 \cdot s]_{q_\ell}$.
- $\underline{\mathtt{C.Add}\left(\mathsf{ct}, \mathsf{ct}'\right)}$. For $\mathsf{ct}, \mathsf{ct}' \in R_{q_\ell}^2$, output $\mathsf{ct}_{\mathtt{add}} \leftarrow \left[\mathsf{ct} + \mathsf{ct}'\right]_{q_\ell}$.
- $\underline{\mathtt{C.Sub}\left(\mathsf{ct}, \mathsf{ct}'\right)}$. For $\mathsf{ct}, \mathsf{ct}' \in R_{q_\ell}^2$, output $\mathsf{ct}_{\mathtt{sub}} \leftarrow \left[\mathsf{ct} - \mathsf{ct}'\right]_{q_\ell}$.
- $\underline{\mathtt{C.Mult}_{\mathsf{evk}}\left(\mathsf{ct}, \mathsf{ct}'\right)}$. For $\mathsf{ct} = (c_0, c_1), \mathsf{ct}' = \left(c_0', c_1'\right) \in \mathcal{R}_{q_\ell}^2$, let $(d_0, d_1, d_2) = \left(c_0 c_0', c_0 c_1' + c_1 c_0', c_1 c_1'\right)$. Output $\mathsf{ct}_{\mathtt{mult}} \leftarrow \left[(d_0, d_1) + \left\lfloor q_L^{-1} \cdot d_2 \cdot \mathsf{evk} \right\rceil\right]_{q_\ell}$.

For more details of the scheme including the correctness and security analysis, we refer the readers to [3].

The above-mentioned scheme deals with message polynomial $\mathfrak{m}$ in some integer polynomial ring $\mathbb{R}$. To encrypt real (or complex) value, HEAAN use a (field) isomorphism $\tau : \mathbb{R}[X] / \left(X^N + 1\right) \rightarrow \mathbb{C}^{N/2}$ called canonical embedding. A plaintext vector $\vec{m} = (m_0, ..., m_{N/2-1})$ is first transformed into $\tau^{-1}(\vec{m}) \in \mathbb{R}[X] / \left(X^N + 1\right)$, and then rounded off to an integer-coefficient polynomial. However, the naive rounding-off $\left\lfloor \tau^{-1}(\vec{m}) \right\rceil$ can derive quite large relative error on the plaintext. To control the error, we round it off after scaling up by p bits for some integer p, i.e., $\left\lfloor 2^p \cdot \tau^{-1}(\vec{m}) \right\rceil$, so that the relative error is reduced. Clearly, a decoding algorithm for $\mathfrak{m}$ would be $2^{-p} \cdot \tau(\mathfrak{m})$:

- $\underline{\mathtt{Ecd}\left(\vec{m}; p\right)}$. For $\vec{m} = (m_0, ..., m_{N/2-1})$ in $\mathbb{C}^{N/2}$ and a precision bit $p > 0$, output a polynomial $\mathfrak{m} \leftarrow \left\lfloor 2^p \cdot \tau^{-1}(\vec{m}) \right\rceil \in R$ where the rounding $\lfloor \cdot \rceil$ is done coefficient-wisely.
- $\underline{\mathtt{Dcd}(\mathfrak{m}; p)}$. For $\mathfrak{m} \in R$, output a plaintext vector $\vec{m}' = 2^{-p} \cdot \tau(\mathfrak{m}) \in \mathbb{C}^{N/2}$.

To sum up, to encrypt a plaintext vector of real (complex) numbers $\vec{m}$, we first encode $\vec{m}$ into $\mathfrak{m} \leftarrow \mathtt{Ecd}\left(\vec{m}; p\right)$ with a certain precision bit p, and then generate a ciphertext $\mathsf{ct} \leftarrow \mathtt{Enc}_{\mathsf{pk}}(\mathfrak{m})$ with the public key pk.

Now consider ct_1 and ct_2 be ciphertexts of $\vec{m}_1$ and $\vec{m}_2$ in $\mathbb{C}^{N/2}$. Since our encoding method scales each plaintext vector up by 2^p, the plaintext vector of a ciphertext $\mathsf{ct}' \leftarrow \mathtt{C.Mult}_{\mathsf{evk}}\left(\mathsf{ct}_1, \mathsf{ct}_2\right)$ is (approximately) $2^p \cdot \vec{m}_1 \odot \vec{m}_2$, not $\vec{m}_1 \odot \vec{m}_2$, which will result in exponential growth of plaintexts. To deal with this problem, we adjust the scaling factor by the following procedure so-called rescaling:

- $\underline{\mathtt{RS}_{\ell \rightarrow \ell'}(\mathsf{ct})}$. For a ciphertext $\mathsf{ct} \in R_{q_\ell}^2$, output $\mathsf{ct}' \leftarrow \left[\lfloor (q_{\ell'}/q_\ell) \cdot \mathsf{ct} \rceil\right]_{q_{\ell'}}$.

After the rescaling procedure $\mathsf{ct}_{\mathtt{mult}} \leftarrow \mathtt{RS}(\mathsf{ct}')$, the plaintext vector of the output $\mathsf{ct}_{\mathtt{mult}}$ is (approximately) $\vec{m}_1 \odot \vec{m}_2$, and the ciphertext modulus q_L is reduced by 2^p. As a result, the level parameter L should be carefully chosen according to the multiplicative depth of a target computation. In order to present our algorithm in a simple form, we will not describe these rescaling procedures for the rest of the paper, but we remark that in the actual use of HEAAN, there should be delicate consideration on scaling of message.

To deal with a plaintext vector of the form $\vec{m} \in \mathbb{C}^k$ having length $K \leq N/2$ for some power-of-two divisor K of $N/2$, HEAAN encrypts $\vec{m}$ into a ciphertext of an $N/2$-dimensional vector $(\vec{m}||\cdots||\vec{m}) \in \mathbb{C}^{N/2}$, where $(\vec{v}||\vec{w})$ denotes the concatenation of two vectors $\vec{v}$ and $\vec{w}$. This implies that a ciphertext of $\vec{m} \in \mathbb{C}^K$ can be understood as a ciphertext of $(\vec{m}||\cdots||\vec{m}) \in \mathbb{C}^{K'}$ for powers-of-two K and K' satisfying $K \leq K' \leq N/2$.

Finally, the HEAAN scheme provides the rotation operation on plaintext slots, i.e., it enables us to securely obtain an encryption of the shifted plaintext vector $(m_r, \ldots, m_{N/2-1}, m_0, \ldots, m_{r-1})$ from an encryption of $(m_0, \ldots, m_{N/2-1})$. It is necessary to generate an additional public information rk, called the rotation key. We denote the rotation operation as follows.

- $\underline{\mathtt{Rot}_{\mathsf{rk}}(\mathsf{ct}; r)}$. For the rotation key rk, output a ciphertext ct' encrypting the (left) rotated plaintext vector of ct by $r(> 0)$ positions as above example. If $r < 0$, it denotes the right rotation by $(-r)$ positions.

We omit a subscript of each algorithm of HEAAN for convenience if it is obvious.

Matrix packing method and rotate function

In this subsection, we describe an encoding method to encrypt a matrix structure in a ciphertext which was also introduced in [8]. Consider an $n \times m$ matrix Z

$$Z = \begin{bmatrix} z_{0,0} & \cdots & z_{0,m-1} \\ \vdots & \ddots & \vdots \\ z_{n-1,0} & \cdots & z_{n-1,m-1} \end{bmatrix}.$$

We first pad zeros to set the number of rows and columns to be powers-of-two, say $\overline{n}$ and $\overline{m}$, and assume that $\log \overline{n} + \log \overline{m} \leq \log(N/2)$. Then we pack the whole matrix in a single ciphertext ct_Z in a column-by-column manner. As described above, the algorithm $\mathtt{Rot}(\mathsf{ct}_Z; r)$ can shift the encrypted vector by r positions. In particular, we can perform row and column rotations of an encrypted matrix with this operation. When $r = \overline{n} \cdot j$, and the result will be the (left) column rotation of the encrypted matrix Z by j columns.

For the row rotation of an encrypted matrix, we use so-called *masking approach*. Consider $n \times m$ matrices M_i and $\overline{M_i}$, where the first $n - i$ rows of M_i (resp. $\overline{M_i}$) are filled with 1 (resp. 0) and the last i rows of M_i (resp. $\overline{M_i}$) are filled with 0 (resp. 1). Let msk_i and $\overline{\mathsf{msk}_i}$ be the ciphertext of M_i and $\overline{M_i}$, respectively.

$$M_i = \begin{bmatrix} 1 & \cdots & 1 \\ 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, \quad \overline{M_i} = \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

To rotate Z by i rows in encrypted state, we first compute $\mathsf{ct}_1 \leftarrow \mathtt{Rot}(\mathsf{ct}_Z, i)$ and $\mathsf{ct}_2 \leftarrow \mathtt{Rot}(\mathsf{ct}_Z, i - n)$. Then, we mask them as $\mathsf{ct}_1 \leftarrow \mathtt{C.Mult}(\mathsf{ct}_1, \mathsf{msk}_i)$ and $\mathsf{ct}_2 \leftarrow \mathtt{C.Mult}\left(\mathsf{ct}_2, \overline{\mathsf{msk}_i}\right)$. As a result, the output of $\mathtt{C.Add}(\mathsf{ct}_1, \mathsf{ct}_2)$ is a ciphertext of upper row rotation of Z by i rows.

Those row and column rotations of an encrypted matrix are denoted as follows:

- $\underline{\mathtt{C.ColumnRot}(\mathsf{ct}_Z, j)}$. For a ciphertext ct_Z of a matrix Z and an integer j, output a ciphertext ct of left column rotation of Z by j columns.
- $\underline{\mathtt{C.RowRot}(\mathsf{ct}_Z, i)}$. For a ciphertext ct_Z of a matrix Z and an integer j, output a ciphertext ct of upper row rotation of Z by i rows.

Semi-parallel GWAS algorithm

A naive application of GWAS analysis can be done by running a logistic regression for each SNP, which resulting in high computational cost since the number of SNPs can be usually hundred thousands or more. To overcome this problem, Sikorska *et al.*[27] proposed a semi-parallel GWAS algorithm which reduces the required computation time from 6 hours to 10-15 minutes using projections.

Let n be the number of samples each of which consists of m (binary) SNP data and k' covariate data. Then the whole SNP data and covariate data can be organized as an $n \times m$ matrix S and an $n \times k'$ matrix X_0, respectively. For $k := k' + 1$, we define a matrix X as the concatenation of a vector whose components are 1 and X_0, denoted by $X = \left(\vec{1}||X_0\right)$. Let $\vec{y}$ be a target binary phenotype vector of length n. With these inputs, the semi-parallel GWAS algorithm outputs the m-dimensional vector $\overrightarrow{\mathsf{pval}}$ which indicates the p-value of each SNP with respect to the target phenotype. The detail of the algorithm is described in Algorithm 1.

The semi-parallel GWAS algorithm involves logistic regression on $(X, \vec{y})$ in the first step, and Fisher Scoring [28] described in Algorithm 2 is one of the most highly efficient algorithm for logistic regression. In both algorithms, $\sigma(x) := 1/(1 + \exp(-x))$ is called sigmoid function. For both algorithms, if the input of

Algorithm 1 The Original Semi-Parallel GWAS

```
Input: SNP matrix $S \in \{0,1\}^{n\times m}$, covariate matrix $X \in \mathbb{Q}^{n\times k}$, phenotype vector $\vec{y} \in \mathbb{Q}^n$, and # iteration iter.
Output: $p$-value vector $\overrightarrow{\mathsf{pval}} = (\mathsf{pval}_1, \cdots, \mathsf{pval}_m) \in \mathbb{Q}^m$.
 1: $(\vec{\beta}, \vec{p}, W) \leftarrow \mathsf{FisherScoring}(X, \vec{y}; \mathsf{iter})$
        $\triangleright$ FisherScoring$(\cdot)$ is described in Algorithm 2
 2: $\vec{v} \leftarrow \log(\vec{p}/(\vec{1}-\vec{p})) + (\vec{y}-\vec{p})/\mathsf{diag}(W)$
 3: $S^* \leftarrow S - X(X^TWX)^{-1}X^TWS$
 4: $\vec{v^*} \leftarrow \vec{v} - X(X^TWX)^{-1}X^TW\vec{z}$
 5: $\vec{c} \leftarrow {S^*}^TW\vec{v^*} \in \mathbb{Q}^m$
 6: $\vec{d} \leftarrow \mathsf{diag}\left({S^*}^TWS^*\right) \in \mathbb{Q}^m$
 7: for $i = 1$ to $m$ do
 8:     $a_i \leftarrow -|c_i/\sqrt{d_i}|$
 9:     $\mathsf{pval}_i = 2\cdot\int_{-\infty}^{a_i}\rho(x)dx \triangleright \rho(x) := \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}x^2\right)$
10: end for
11: return $\overrightarrow{\mathsf{pval}}$
```

functions such as logarithm (log), division (/) and sigmoid (σ) is a vector, then it means to apply the function component-wisely resulting in the output vector of the same length. The notation T in the superscript denotes the matrix transpose. The operation $\odot$ denotes the Hadamard (component-wise) multiplication of two vectors, and the notation $\mathsf{diag}(\cdot)$ with an input of a square matrix means the diagonal vector of the input.

Our optimization methodology

The aim of this study is to construct an HE algorithm for privacy-preserving semi-parallel GWAS computation. Since non-polynomial operations such as matrix inverse or real number inversion is a challenging stuff in HE, we need to modify the original semi-parallel GWAS algorithm into HE-friendly form for efficiency. Moreover, the super-large data size of GWAS requires too much computational cost in encrypted state, and this issue should be resolved. In this regard, we introduce our optimization methodology to the algorithm.

Algorithm 2 FisherScoring

```
Input: Covariate matrix $X \in \mathbb{Q}^{n\times k}$, phenotype vector $\vec{y} \in \mathbb{Q}^n$, and # iteration iter.
Output: Coefficient vector $\vec{\beta} \in \mathbb{Q}^k$, fitted vector $\vec{p} \in \mathbb{Q}^n$, weight matrix $W \in \mathbb{Q}^{n\times n}$
 1: $\vec{\beta}^{(0)} = \vec{0} \in Q^k$, $\vec{p}^{(0)} = \overrightarrow{0.5} \in \mathbb{Q}^n$, $W^{(0)} = 0.25 \cdot I_n$
 2: for $t = 0$ to iter $- 1$ do
 3:     $\vec{v}^{(t)} \leftarrow \log\left(\vec{p}^{(t)}/(\vec{1}-\vec{p}^{(t)})\right) + \left(\vec{y}-\vec{p}^{(t)}\right)/\mathsf{diag}\left(W^{(t)}\right)$
 4:     $\vec{\beta}^{(t+1)} \leftarrow \left(X^TW^{(t)}X\right)^{-1}X^TW^{(t)}\vec{v}^{(t)}$
 5:     $\vec{p}^{(t+1)} \leftarrow \sigma\left(X\vec{\beta}^{(t+1)}\right)$
 6:     $W^{(t+1)} \leftarrow$ diagonal matrix of $\vec{p}^{(t+1)} \odot \left(\vec{1}-\vec{p}^{(t+1)}\right)$
 7: end for
 8: return $(\vec{\beta}^{(\mathsf{iter})}, \vec{p}^{(\mathsf{iter})}, W^{(\mathsf{iter})})$
```

Modification of fisher scoring

The main obstacle of Fisher Scoring (Algorithm 2) is a matrix inversion for $U = X^TWX \in \mathbb{Q}^{k\times k}$. To overcome this problem, we exploit the adjoint matrix $\mathsf{adj}(U)$ and the determinant $\mathsf{det}(U)$ of U. For (i,j)-minor $M_{i,j} \in \mathbb{Q}$ of U, $\mathsf{adj}(U)$ and $\mathsf{det}(U)$ are obtained from basic linear algebra:

$$\mathsf{adj}(U) = \left[(-1)^{i+j}\cdot M_{i,j}\right] \in \mathbb{Q}^{k\times k}, \qquad (1)$$

$$\mathsf{det}(U) = \sum_{i=0}^{k-1} u_{i,0}\cdot(-1)^i\cdot M_{i,0}. \qquad (2)$$

We express the inverse matrix U^{-1} as $U^{-1} = \frac{1}{\mathsf{det}(U)}\cdot \mathsf{adj}(U)$. To be precise, observe that

$$\begin{aligned}\vec{v}^{(t)} &= \log\left(\frac{\vec{p}^{(t)}}{\vec{1}-\vec{p}^{(t)}}\right) + \frac{\vec{y}-\vec{p}^{(t)}}{\mathsf{diag}\left(W^{(t)}\right)}\\ &= X\vec{\beta}^{(t)} + \frac{\vec{y}-\vec{p}^{(t)}}{\mathsf{diag}\left(W^{(t)}\right)},\end{aligned}$$

from which we obtain an iterative updating equation on $\vec{\beta}^{(t)}$ as follows:

$$\begin{aligned}\vec{\beta}^{(t+1)} &= U^{-1}X^TW^{(t)}\left(X\vec{\beta}^{(t)} + \frac{\vec{y}-\vec{p}^{(t)}}{\mathsf{diag}\left(W^{(t)}\right)}\right)\\ &= \vec{\beta}^{(t)} + U^{-1}X^T\left(\vec{y}-\vec{p}^{(t)}\right)\\ &= \vec{\beta}^{(t)} + \frac{1}{\mathsf{det}(U)}\cdot\mathsf{adj}(U)X^T\left(\vec{y}-\vec{p}^{(t)}\right).\end{aligned}$$

Here one needs to compute the inverse of $\mathsf{det}(U)$, but this non-polynomial operation is rather expensive in HE. The key observation on this equation is that the second term $U^{-1}X^T\left(\vec{y}-\vec{p}^{(t)}\right)$ essentially converges to 0 as $t \to \infty$ since $\vec{\beta}^{(t)}$ converges to some point. From this, we may expect that the convergence would be still valid even when we neglect the term $\mathsf{det}(U)^{-1}$ and substitute it by some appropriate constant. Namely, we can modify the equation as

$$\vec{\beta}^{(t+1)} = \vec{\beta}^{(t)} + \alpha\cdot\mathsf{adj}(U)X^T\left(\vec{y}-\vec{p}^{(t)}\right).$$

for some constant $\alpha > 0$. In practice, this approximate version of the Fisher scoring algorithm works quite well with slightly slower convergence rate.

Computing diag $\left({S^*}^TWS^*\right)$ without S^*

The main observation of this subsection is that computation of $n\times m$ matrix S^* in Algorithm 1 is superfluous for obtaining p-values. Rather than computing S^*, we directly compute $\mathsf{det}(U)\cdot\mathsf{diag}\left({S^*}^TWS^*\right)$, which can be obtained without computing (matrix) inversion. To be precise, using the fact that $S^* = S - XU^{-1}V$ for $V := X^TWS$, we get

$$\begin{aligned} S^{*T}WS^* &= \left(S - XU^{-1}V\right)^T W\left(S - XU^{-1}V\right) \\ &= S^TWS - V^TU^{-1}V. \end{aligned}$$

Based on this observation, we compute $\mathsf{det}(U) \cdot \mathsf{diag}\left(S^{*T}WS^*\right)$ by following:

1 Compute $U = X^TWX$ and $V = X^TWS$.
2 Compute $\mathsf{adj}(U)$ and $\mathsf{det}(U)$.
3 Compute $\mathsf{det}(U)\cdot\mathsf{diag}\left(S^TWS\right) - \mathsf{diag}\left(V^T\mathsf{adj}(U)V\right)$.

Approximate computation of $S^{*T}W\vec{v}^*$

We also take the main observation of the previous subsection so that we do not compute S^*. From the definition of S^* and $\vec{v}^*$, it holds that $S^{*T}W\vec{v}^* = S^TW\vec{v}^*$. Then we have the following equations:

$$\begin{aligned} S^TW\vec{v}^* &= S^TW\left(I - XU^{-1}X^TW\right)\vec{v} \\ &= S^TW\left(I - XU^{-1}X^TW\right)\frac{\vec{y}-\vec{p}}{\mathsf{diag}(W)} \\ &= S^T\left(\vec{y}-\vec{p}\right) - S^TWXU^{-1}X^T\left(\vec{y}-\vec{p}\right) \\ &\simeq S^T\left(\vec{y}-\vec{p}\right) \end{aligned}$$

where the last approximation is valid since the term $X^T\left(\vec{y}-\vec{p}\right)$ is sufficiently close to the zero vector, which is resulted from the Fisher Scoring. For example, each component of $X^T\left(\vec{y}-\vec{p}\right)$ was approximately 10^{-30} in our experiment with 4 iterations in Fisher scoring. Refer to "Dataset description" section for the specific description of datasets. Therefore, we compute $\mathsf{det}(U) \cdot S^T\left(\vec{y}-\vec{p}\right)$ which is a reliable approximation of $\mathsf{det}(U) \cdot S^{*T}W\vec{v}^*$, in much less computational costs.

Our modified semi-parallel GWAS algorithm

To sum up all our algorithmic optimization techniques described in above, we have Algorithm 3 and 4, which are modified Fisher Scoring and semi-parallel GWAS algorithms, respectively.

In Algorithm 3, the constant α takes a similar role to the learning rate in gradient descent algorithm [29], which can be adjusted if necessary.

We remark that step 8 of Algorithm 4, a conversion procedure from the squared z-score z_i to the p-value pval_i, is done in unencrypted state. Namely, we decrypt the ciphertext of z_i for $1 \le i \le m$ after step 7 so that z_i's are publicized. We stress that the squared z-score has exactly *the same information* as the p-value, so publishing squared z-scores does not leak any additional information more than publishing p-values.

Algorithm 3 ModifiedFisherScoring

```
Input: Covariate matrix $X \in \mathbb{Q}^{n\times k}$, phenotype vector $\vec{y} \in \mathbb{Q}^n$, # iteration iter, and constant $\alpha > 0$.
Output: Coefficient vector $\vec{\beta} \in \mathbb{Q}^k$, fitted vector $\vec{p} \in \mathbb{Q}^n$, weight matrix $W \in \mathbb{Q}^{n\times n}$
1: $\vec{\beta}^{(0)} = \vec{0} \in Q^k$, $\vec{p}^{(0)} = \vec{0.5} \in \mathbb{Q}^n$, $W^{(0)} = 0.25 \cdot I_n$
2: for $t = 0$ to iter $- 1$ do
3:     $U^{(t)} \leftarrow X^TW^{(t)}X$
4:     $\vec{\beta}^{(t+1)} \leftarrow \vec{\beta}^{(t)} + \alpha \cdot \mathsf{adj}\left(U^{(t)}\right)X^T\left(\vec{y} - \vec{p}^{(t)}\right)$
5:     $\vec{p}^{(t+1)} \leftarrow \sigma\left(X\vec{\beta}^{(t+1)}\right)$
6:     $W^{(t+1)} \leftarrow$ diagonal matrix of $\vec{p}^{(t+1)} \odot \left(\vec{1} - \vec{p}^{(t+1)}\right)$
7: end for
8: return $\vec{\beta}^{(\mathsf{iter})}, \vec{p}^{(\mathsf{iter})}, W^{(\mathsf{iter})}$
```

Homomorphic evaluation of the modified semi-parallel GWAS algorithm

Upon HE-friendly algorithms discussed in the previous section, there still remain computational issues regarding more fundamental operations. Recall that HEAAN basically supports component-wise addition and multiplication along with data slot rotations. However, we encrypt the data matrix by column-by-column manner, and our algorithms include complex operations such as matrix multiplication, evaluation of the adjoint matrix, a sigmoid function, and so on. In this regard, we specify how we can deal with such operations efficiently, reducing the number of multiplications or the total depth of multiplications required which are the main bottleneck of HE.

Note that this section consists of rather technical contents related to HE, since it includes HE algorithms of all building blocks for Algorithm 3 and Algorithm 4. One can simply embrace the fact that every operation required in

Algorithm 4 Modified Semi-Parallel GWAS

```
Input: SNP matrix $S \in \{0,1\}^{n\times m}$, covariate matrix $X \in \mathbb{Q}^{n\times k}$, phenotype vector $\vec{y} \in \mathbb{Q}^n$, # iteration iter, and constant $\alpha > 0$.
Output: $p$-value vector $\overrightarrow{\mathsf{pval}} = \left(\mathsf{pval}_1, \cdots, \mathsf{pval}_m\right) \in \mathbb{Q}^m$.
1: $\vec{\beta}, \vec{p}, W \leftarrow$ ModifiedFisherScoring $(X, \vec{y}; \mathsf{iter}, \alpha)$
2: $U \leftarrow X^TWX$, and compute $\mathsf{adj}(U), \mathsf{det}(U)$
3: $V \leftarrow X^TWS$
4: $\vec{c} \leftarrow S^T\left(\vec{y} - \vec{p}\right)$
5: $\vec{d} \leftarrow \mathsf{det}(U) \cdot \mathsf{diag}\left(S^TWS\right) - \mathsf{diag}\left(V^T\mathsf{adj}(U)V\right)$
6: for $i = 1$ to $m$ do
7:     $z_i \leftarrow \mathsf{det}(U) \cdot c_i^2/d_i$
8:     $\mathsf{pval}_i = 2 \cdot \int_{-\infty}^{-\sqrt{z_i}} \rho(x)dx$   ▷ Done in unencrypted state
9: end for
10: return $\overrightarrow{\mathsf{pval}}$
```

Algorithm 3 and Algorithm 4 can be efficiently done based on HE, if not really interested in the details.

Hereafter, $[a]_k$ with an integer a denotes a residue number in $[0, k-1]$ modulo k. An n-dimensional vector $\vec{a} = (a_1, \cdots, a_n)$ is simply denoted by $(a_i)_{1\le i\le n}$, and an $n \times m$ matrix A having (i,j)-entry $a_{i,j}$ is denoted by $[a_{i,j}]_{1\le i\le n, 1\le j\le m}$. For both cases, if the size is obvious from context, we simply write a vector by (a_i), and a matrix by $[a_{i,j}]$. Every vector and matrix in this section is assumed to be of size power-of-two, which is in line with our packing method introduced in "Matrix packing method and rotate function" section.

Adjoint matrix and determinant

In step 4 of Algorithm 3 and step 2 of Algorithm 4, we need to compute the adjoint matrix and the determinant of the matrix $U = [u_{i,j}]_{i,j} = X^T WX$.

We recall Eqs. 1 and 2 for adjoint matrix and determinant. Given an encryption of U, denoted by C_U, we generate $(k-1)^2$ ciphertexts $C_{i,j}$ for $1 \le i,j \le k-1$ from C_U, whose plaintext is an i-row (upper) rotation and j-column (left) rotation of U. We first consider the 0-th plaintext slot, i.e., the $(0,0)$-position of the plaintext matrix, of the ciphertexts. Since every $u_{a,b}$ for $1 \le a, b \le k-1$ is a $(0,0)$-entry of plaintext matrix for one and only one ciphertext $C_{a,b}$, we can compute a ciphertext whose $(0,0)$-entry of the plaintext matrix is $M_{0,0}$ from $C_{a,b}$'s, by homomorphically evaluating the polynomial f which outputs $M_{0,0}$ with input $u_{a,b}$'s.

Now observe that $M_{a,b}$ and $M_{a',b'}$ have a formula of the same form where the subscript indices of $u_{i,j}$ are shifted by $(a'-a, b'-b)$ modulo k. Thanks to this index-shifting property, the homomorphic evaluation of the polynomial f with input $C_{a,b}$'s essentially outputs a ciphertext of which the plaintext matrix is $[M_{i,j}]$.

After computing the ciphertext of $[M_{i,j}]$ as above, we can obtain the ciphertext C_{adj} of $\mathsf{adj}(U)$ by multiplying a ciphertext C_{sgn} of $\left[(-1)^{i+j}\right]$. Finally, the ciphertext C_{det} of determinant $\mathsf{det}(U)$ is easily obtained from the homomorphic multiplication of C_U and C_{adj}, followed by $\log k$ rotations and summations.

In case of $k = 4$, for example, the polynomial f is defined as $f\left([u_{i,j}]_{1\le i,j\le 3}\right) = u_{1,1}u_{2,2}u_{3,3} - u_{1,1}u_{2,3}u_{3,2} - u_{2,1}u_{1,2}u_{3,3} + u_{2,1}u_{1,3}u_{3,2} + u_{3,1}u_{1,2}u_{2,3} - u_{3,1}u_{1,3}u_{2,2}$. Then, the homomorphic evaluation to obtain C_{adj} is done as Algorithm 5.

Algorithm 5 `C.Adj` $(C_U; C_{\mathsf{sgn}})$ for $k = 4$

```
Input: Ciphertexts $C_U$ of $U$ and $C_{\mathsf{sgn}}$ of $\left[(-1)^{i+j}\right]_{0\le i,j\le 3}$
Output: Ciphertext $C_{\mathsf{adj}}$ of $\mathsf{adj}(U)$
1:  for $i = 1$ to $3$ do
2:      $C_{i,0} \leftarrow$ C.RowRot$(C_U, i)$
3:      for $j = 1$ to $3$ do
4:          $C_{i,j} \leftarrow$ C.ColumnRot$(C_{i,0}, j)$
5:      end for
6:  end for
7:  $C_1 \leftarrow$ C.Mult(C.Mult$(C_{1,1}, C_{2,2})$, $C_{3,3}$)
8:  $C_2 \leftarrow$ C.Mult(C.Mult$(C_{1,1}, C_{2,3})$, $C_{3,2}$)
9:  $C_3 \leftarrow$ C.Mult(C.Mult$(C_{2,1}, C_{1,2})$, $C_{3,3}$)
10: $C_4 \leftarrow$ C.Mult(C.Mult$(C_{2,1}, C_{1,3})$, $C_{3,2}$)
11: $C_5 \leftarrow$ C.Mult(C.Mult$(C_{3,1}, C_{1,2})$, $C_{2,3}$)
12: $C_6 \leftarrow$ C.Mult(C.Mult$(C_{3,1}, C_{1,3})$, $C_{2,2}$)
13: for $i = 2$ to $6$ do
14:     if $i$ is even then
15:         $C_1 \leftarrow$ C.Sub$(C_1, C_i)$
16:     else
17:         $C_1 \leftarrow$ C.Add$(C_1, C_i)$
18:     end if
19:     $C_{\mathsf{adj}} \leftarrow$ C.Mult$(C_1, C_{\mathsf{sgn}})$
20: end for
21: return $C_{\mathsf{adj}}$
```

Matrix multiplications

Let $A = [a_{i,j}]$ be an $n \times k$ matrix with $n \ge k$ and $B = [b_{i,j}]$ be an $n \times m$ matrix. We use an Algorithm 6 computing a ciphertext C_{A^TB} of A^TB from ciphertexts C_A and C_B of A and B, which is inspired from the hybrid method by Juvekar et al. [30].

As the first step, we compute k ciphertexts of

$$\mathsf{diag}_t(A) = \left(a_{i,[i-t]_k}\right)_{0\le i\le n-1}$$

for $0 \le t \le k-1$. For this, we use ciphertexts dmsk_t of $n \times k$ masking matrix, of which the (i,j)-entry is $\delta_{[i+t]_k,j}$. Here $\delta_{i,j}$ denotes the Kronecker Delta. By summing column rotations of $A \odot \mathsf{dmsk}_t$, we get ciphertexts of $n \times m$ matrices $\mathsf{Expdiag}_t(A)$ having m identical columns $\mathsf{diag}_t(A)$. Then, we compute the following matrix M:

$$\begin{aligned} M &= \sum_{t=0}^{k-1} \rho_t\left(\mathsf{Expdiag}_t(A) \odot B\right) \\ &= \sum_{t=0}^{k-1} \rho_t\left(\left[a_{i,[i-t]_k} \cdot b_{i,j}\right]\right) \\ &= \sum_{t=0}^{k-1} \left[a_{[i+t]_n,[i]_k} \cdot b_{[i+t]_n,j}\right] \in \mathbb{Q}^{n\times m}, \end{aligned}$$

where ρ_t is an (upward) t-rotation of matrix by row. Thus, by properly summing rows of M, we obtain $A^TB = \sum_{t=0}^{n-1}[a_{t,i} \cdot b_{t,j}] \in \mathbb{Q}^{k\times m}$. The detail of this algorithm is described in Algorithm 6.

Indeed, our GWAS algorithm contains several matrix multiplications of the form A^TDB for the diagonal matrix D. For this, we first compute $B' = DB$ by $\mathsf{Expdiag}_0(D) \odot B$, and then obtain A^TDB by computing A^TB' with the

Algorithm 6 `C.MatMul`(C_A, C_B)

```
Input: Ciphertexts $C_A$ of $A \in \mathbb{Q}^{n\times k}$ and $C_B$ of $B \in \mathbb{Q}^{n\times m}$ with $n \geq k$, and masking ciphertexts $\{\mathsf{dmsk}_t\}_t$
Output: A ciphertext of $A^TB$.
1: $C_M \leftarrow 0$
2: for $t = 0$ to $k-1$ do
3:     $C_t \leftarrow$ C.Mult $(\mathsf{dmsk}_t, C_A)$
4:     for $i = 0$ to $\log k - 1$ do
5:         $C_t \leftarrow$ C.Add $\left(C_t, \text{C.ColumnRot}\left(C_t, 2^i\right)\right)$
6:     end for
7:     $C_M \leftarrow$ C.Add $(C_M$, C.RowRot (C.Mult $(C_t, C_B)$ , $t))$
8: end for
9: for $i = 0$ to $\log(n/k) - 1$ do
10:     $C_M \leftarrow$ C.Add $\left(C_M, \text{C.RowRot}\left(C_M, 2^i \cdot k\right)\right)$
11: end for
12: return $C_M$
```

above method. Note that this requires only one additional Hadamard multiplication.

Matrix-vector multiplications

By understanding a vector by an one column matrix, we can perform all matrix-vector multiplications in our algorithms, except $X\vec{\beta}$ that appears in step 5 of Algorithm 3.

In fact, we can also compute $X\vec{\beta}$ by changing Algorithm 6 a little bit. Recall that $X = [x_{i,j}]$ is an $n \cdot k$ matrix and $\vec{\beta} = (\beta_i)$ is a k-length vector, and it holds that $n \geq k$. We now again compute k ciphertexts of $\mathsf{diag}_t(X)$, and then compute

$$\sum_{t=0}^{k-1} \mathsf{diag}_t(X) \odot \rho_{-t}\left(\left(\vec{\beta}||\cdots||\vec{\beta}\right)\right)$$
$$= \sum_{t=0}^{k-1}\left(x_{i,[i-t]_k}\right) \odot \left(\beta_{[i-t]_k}\right)$$
$$= \sum_{t=0}^{k-1}\left(x_{i,[i-t]_k} \cdot \beta_{[i-t]_k}\right) = X\vec{\beta},$$

whose algorithm is described by Algorithm 7.

We remark that there is another simple method for matrix-vector multiplication that we use for $\vec{c} = S^T\left(\vec{y} - \vec{p}\right)$. For simplicity, let $\vec{x} = (x_i) := \vec{y} - \vec{p}$. Then by rotating and summing all rows of $S \odot \left[\vec{x}||\cdots||\vec{x}\right] = \left[s_{i,j} \cdot x_i\right]$, we obtain a matrix having the same size with S and consisting of identical *rows* $\vec{c} = S^T\vec{x}$. This requires only one Hadamard multiplication and $\log n$ rotations. However, strictly speaking, this resulting ciphertext is not a ciphertext of $S^T\vec{x}$, since it encrypts a matrix having $\vec{c}$ row-wisely, not column-wisely. Thus we can only use this simple method only for $S^T\vec{x}$, where this row-wise packing does not matter after then. The detail of this algorithm is described in Algorithm 8.

Algorithm 7 `C.MatVecMul`(C_X, C_β)

```
Input: Ciphertexts $C_X$ of $X \in \mathbb{Q}^{n\times k}$ and $C_{\vec{\beta}}$ of $\vec{\beta} \in \mathbb{Q}^k$ and masking ciphertexts $\{\mathsf{dmsk_t}\}_t$
Output: A ciphertext of $X\beta$.
1: $C_M \leftarrow 0$
2: for $t = 0$ to $k-1$ do
3:     $C_t \leftarrow$ C.Mult$(\mathsf{dmsk}_t, C_X)$
4:     for $i = 0$ to $\log k - 1$ do
5:         $C_t \leftarrow$ C.Add $\left(C_t, \text{C.ColumnRot}\left(C_t, 2^i\right)\right)$
6:     end for
7:     $C_M \leftarrow$ C.Add $\left(C_M, \text{C.Mult}\left(C_t, \text{C.RowRot}\left(C_{\vec{\beta}}, -t\right)\right)\right)$
8: end for
9: return $C_M$
```

Fast diag $\left(A^TBA\right)$ computations

To obtain $\mathsf{diag}\left(A^TBA\right)$, one can perform matrix multiplication followed by diagonal extraction, but it is obviously not optimal since this computes unnecessary entries of A^TBC other than diagonal entries. Thus we use another method that only compute the diagonal entries.

Let $A = \left[a_{i,j}\right]$ be an $n \times m$ matrix. As an incremental step, we first consider $\mathsf{diag}\left(A^TDC\right)$ where $D = \left[d_{i,j}\right]$ is an $n \times n$ diagonal matrix and $C = \left[c_{i,j}\right]$ is an $n \times m$ matrix. Then it holds that $\mathsf{diag}\left(A^TDC\right)_j = \sum_{i=0}^{n-1} d_{i,i} \cdot a_{i,j} \cdot c_{i,j}$. Now, from an encryption of D, we compute an encryption of $n \times m$ matrix $\mathsf{Expdiag}_0(D)$ and then by rotating and summing

$$A \odot \mathsf{Expdiag}_0(D) \odot C = \left[d_{i,i} \cdot a_{i,j} \cdot c_{i,j}\right]_{i,j}$$

through all rows, we obtain a matrix consisting of identical rows $\mathsf{diag}\left(A^TDC\right)$. One can easily check that Algorithm 8 with input C_A and C.Mult $\left(C_{\mathsf{diag}(D)}, C_C\right)$ exactly performs this computation, and then we omit the explicit algorithm. Note that this can be directly applied for $\mathsf{diag}\left(S^TWS\right)$ computation of step 5 of Algorithm 4.

Algorithm 8 `C.MatVecMul`$_2(C_S, C_{\vec{x}})$

```
Input: Ciphertexts $C_S$ of a matrix $S \in \mathbb{Q}^{n\times m}$ and $C_{\vec{x}}$ of a vector $\vec{x} \in \mathbb{Q}^n$
Output: A ciphertext of a matrix having identical rows $S^T\vec{x}$.
1: $C_M \leftarrow$ C.Mult $(C_S, C_{\vec{x}})$
2: for $i = 0$ to $\log n - 1$ do
3:     $C_M \leftarrow$ C.Add $\left(C_M, \text{C.RowRot}\left(C_M, 2^i\right)\right)$
4: end for
5: return $C_M$
```

Toward our goal $\mathsf{diag}\left(A^TBA\right)$ with a full matrix B, we exploit the above diagonal-case method after decomposing B into diagonal matrices. Let B_t be a diagonal matrix with the diagonal $\mathsf{diag}_t(B)$ for $0 \le t \le n-1$, then it holds that

$$\mathsf{diag}\left(A^TBA\right) = \sum_{t=0}^{n-1} \mathsf{diag}\left(A^T \cdot B_t \cdot \rho_t(A)\right).$$

Therefore, after obtaining encryptions of $\mathsf{Expdiag}_t(B)$ and $\rho_t(A)$ from encryptions of B and A, we can directly apply the diagonal-case method on each $\mathsf{diag}(A^T \cdot B_t \cdot \rho_t(A))$ computation for $1 \le t \le n$ and finally obtain the encryption of $\mathsf{diag}\left(A^TBA\right)$.

Here we again remark that, since these methods use Algorithm 8, they also ruin the column-wise packing as we already pointed out. Hence after applying these methods, it would be hard to perform another matrix operation. Indeed, one can check that the diagonal extractions are required for step 5 of Algorithm 4, which is the last part of algorithm that uses matrix structure.

Approximate computation of sigmoid

Since the sigmoid function $\sigma(x) = 1/(1+\exp(-x))$ is not a polynomial, we exploit an approximate polynomial of the function to evaluate based on HE. Following the methodology of [8, 22], we used least square approximation method over the interval $[-8, 8]$. The approximate polynomials $g(x)$ of degree 7 is computed as

$$0.5+1.735\cdot\frac{x}{8}-4.194\cdot\left(\frac{x}{8}\right)^3+5.434\cdot\left(\frac{x}{8}\right)^5-2.507\cdot\left(\frac{x}{8}\right)^7.$$

The maximal error between $\sigma(x)$ and $g(x)$ is approximately 0.032.

Inverse of real numbers

In step 7 of Algorithm 4, we need to compute the inverse of d_i for $1 \le i \le m$. To compute the inverse of real numbers, we exploit the Goldschmidt's division algorithm [31], which outputs an approximate value of the inverse through iterative polynomial evaluations. Refer to [32] for more details of the algorithm.

Results

In this section, we present the experimental results of our modified semi-parallel GWAS algorithm based on HEAAN with a publicly available library [5]. All experiments were implemented in C++ 11 standard, and performed on Linux with Intel Xeon CPU E5-2620 v4 at 2.10GHz processor with multi-threading (8 threads) turned on.

Dataset description

We used a dataset of 245 samples where each sample contains a binary phenotype, 3 covariates (height, weight, age), and 25,484 SNP data provided by iDASH 2018 competition. The dataset is divided into two sets named by iDash_Test and iDash_Eval each composed of 245 samples containing common phenotype and 3 covariates but different number of SNPs; 10,643 and 14,841 SNPs, respectively. We used iDash_Test to set optimal parameters, and iDash_Eval was used to evaluate our algorithm in the competition. Note that the first column of the covariate matrix $X \in \mathbb{Q}^{n\times k}$ is a vector of which all the components are 1. Therefore, the parameters are $(n, m, k) = (245, 10643, 4)$ for iDash_Test and $(n, m, k) = (245, 14841, 4)$ for iDash_Eval.

Experimental setting and parameter selection

We propose two HEAAN parameter sets achieving 128-bit or higher security for two experiments denoted by Exp I and Exp II in Table 1. The security levels of HEAAN parameter sets were estimated with Albrecht's security estimator [25, 26] of which inputs are the ring dimension N, the modulus Q, the Hamming weight h of a secret polynomial, and the error distribution $\chi_{\mathtt{err}}$. Note that since the modulus of the evaluation key evk is 2^{2L}, the security of HEAAN is estimated with input $\left(N, Q = 2^{2L}, \mathsf{h}, \chi_{\mathtt{err}}\right)$.

Exp I is a streamlined version operating Algorithm 4 only until step 5; that is, it does not perform the last division process. Note that Exp II that includes the division process (step 7 of Algorithm 4) naturally requires larger L than Exp I.

We first set the scaling parameter p to be sufficiently large so that errors derived from HEAAN do not effect on significant bits of plaintexts. The level parameter L is chosen by considering the fact that p levels are consumed for each homomorphic multiplication. See "Approximate homomorphic encryption scheme HEAAN" section for specific definitions of HEAAN parameters. Besides thoese HEAAN parameters, one also needs to select an appropriate constant $\alpha > 0$ in Algorithm 3. After experimenting on several values, we set $\alpha = 8$. Note that the choice of α merely depends on the size of X, but not on S.

Experimental results and evaluation

We demonstrated our modified semi-parallel GWAS algorithm in encrypted state, and evaluated the accuracy

Table 1 Parameters for HEAAN, and running time of `KeyGen`, `Enc` and `Dec`

Exp	HE params				Time (sec)		
	log N	L	p	h	KeyGen	Enc	Dec
I	17	1300	50	56	157	68	0.28
II	17	1700	50	78	197	90	0.15

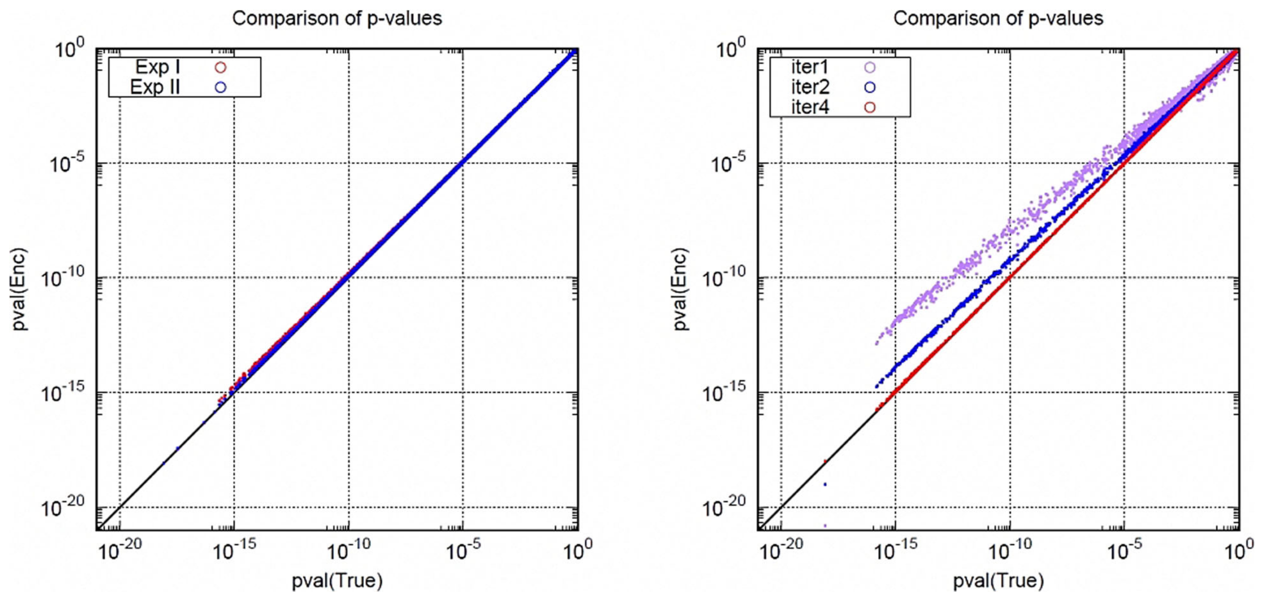

Fig. 1 Comparison of p-values on iDASH datasets: **a** the left figure represents Exp I for iter = 1, 2, 4 on iDash_Test, **b** the right figure represents Exp I and Exp II for iter = 4 on iDash_Eval

of our algorithm comparing it to that of the original algorithm which is performed in unencrypted state. The comparison result of p-value is described as a (log-scale) graph in Fig. 1. We plotted each SNPs according to the p-values computed by original algorithm and ours denoted by True and Enc, respectively. The diagonal line represents the line $y = x$, and closer distribution of points to this line implies higher accuracy. The Fig. 1a shows that the accuracy of our algorithm increases with the number of iterations for Fisher scoring, where the data set iDash_Test is used. The Fig. 1b shows that the accuracy of Exp II, which includes a division procedure, is comparable to that of Exp I without the division, where the data set iDash_Eval is used. Comparing Exp I and Exp II, there exists a trade-off between computational time and information leakage. The output of Exp I is the vector of squared statistics $(z_i)_{1\leq i\leq m}$ which has exactly same information with the p-value vector pval, but it takes 20 minutes longer than Exp I. On the other hand, since Exp I outputs the numerator $\mathsf{det}(U) \cdot c_i^2$ and the denominator d_i (in Algorithm 4) separately, it leaks some information more than p-values. However, it still seems to be very hard to extract any important information of input data from the numerator and denominator.

For more concrete evaluation, we classified each SNP as positive or negative depending on whether the corresponding p-value is larger or smaller than the given threshold (e.g. 10^{-2}, 10^{-5}, or 10^{-12}). Then, the accuracy of our algorithm compared to the original algorithm can be checked by a well-known statistical measure called F_1 score. The F_1 score of our algorithm is calculated by regarding the positive SNPs classified by the original GWAS algorithm to be the correct positive samples. For the formal definition of F_1 score, we refer readers to [33].

The performance of our algorithm including the computation time and the F_1 score on each parameter set is described in Table 2, where iter denotes the number of iterations in Fisher Scoring, Comp. time denotes the running time of our algorithm in encrypted state, and TH denotes the threshold of p-values for classification.

As we have seen in Fig. 1, more iterations of Fisher scoring provides higher accuracy measured by higher F_1 score. Note that 4 iterations suffice to provide high F_1 score even in a very small threshold such as 10^{-12}. Also, Exp II calculating approximate inverse in encrypted state provides almost similar F_1 score to Exp I without such approximation. It implies that the error of inverse approximation does not seriously impact the whole approximation. Furthermore, Exp II shows even higher F_1 score than Exp I due to the cancellation of errors from our algorithmic approximation and that from the inverse approximation.

For about 15,000 SNP data, our algorithm works in less than 40 minutes when we exclude step 7 of Algorithm 4 in encrypted state, or in about 60 minutes otherwise. We emphasize that each iteration of Fisher scoring takes about 3 minutes while the Goldschmidt's division algorithm takes less than 30 seconds. Exp II takes much longer time than Exp I due to the larger level parameter L.

Table 2 Experimental results for each parameter set

Data	Params		Comp. time	Memory	F_1 Score		
	Exp	iter		(GB)	TH: 10^{-2}	TH: 10^{-5}	TH: 10^{-12}
iDash_Test	I	1	13 min*	9.3	0.960	0.969	0.243
	I	2	27 min	16.7	0.985	0.985	0.955
	I	**4**	**32 min**	**16.7**	**1.000**	**0.999**	**0.997**
	II	**4**	**52 min**	**22.0**	**1.000**	**0.999**	**0.998**
iDash_Eval	**I**	**4**	**38 min**	**19.4**	**0.998**	**0.995**	**0.992**
	II		**62 min**	**25.4**	**0.998**	**0.996**	**0.994**

*: We used more streamlined parameter; $\log N = 16, L = 950, p = 50, \mathrm{h} = 91$

Discussion

Scalability

Our algorithm is executed and evaluated with about hundreds of samples each containing ten thousand SNPs, and 3 covariates which can be seen as a small-size data in usual GWAS analysis. We emphasize that our algorithm is highly scalable in the number of samples or SNPs, since we circumvent the naive execution of large-sized matrix operations through the proper algorithmic modification. To test the scalability of our algorithm in practice, we randomly generated 500 samples each of which consist of 3 covariates and 30,000 SNPs. Each column of the covariate matrix was uniform randomly generated in the interval $[150, 200]$, $[40, 100]$ and $[20, 80]$ considering height, weight and age, respectively. Each element of the SNP matrix was uniform randomly chosen as a binary matrix. The experiment $\mathsf{Exp}\,1$ on this random dataset encrypted with properly chosen hyperparameters $\mathsf{iter} = 4$ and $\alpha = 2^{-7}$ still showed quite accurate p-value result compared to the result obtained by running Algorithm 2 in unencrypted state within 2 h.

Fisher Scoring

Our HE-friendly modified Fisher scoring (Algorithm 3) works quite well in practice, but there still remains to obtain some theoretical results on the convergence of the algorithm with respect to the new parameter α. Furthermore, we should consider an error in every operation derived from HEAAN when homomorphically evaluate the algorithm. As a result, research on the convergence of the *erroneous* version of our modified Fisher scoring algorithm should be very interesting topic as a further work. In addition, we note that our modified Fisher scoring algorithm can be generally used for logistic regression, not restricted to GWAS algorithm.

Conclusions

Interest on privacy-preserving genome data analysis based on HE has grown up very rapidly since the annual iDASH competition was launched, and GWAS is one of the most important technologies in this area which was also selected as one of three tasks in iDASH 2018 competition. Our HE-friendly modified semi-parallel GWAS algorithm was successfully implemented based on an approximate HE scheme HEAAN, and we could obtain the p-value result in about 30–40 minutes for 10,000–15,000 SNP data with sufficiently high accuracy compared to the result obtained in unencrypted state.

Abbreviations

HE: Homomorphic encryption; SNP: Single nucleotide polymorphism; GWAS: Genome-wide association study; TH: threshold

Acknowledgements

We thank Kyoohyung Han for the help on finding an error point of our code implementation. We also thank Minki Hhan for giving us helpful discussion on designing efficient algorithms.

About this supplement

This article has been published as part of *BMC Medical Genomics Volume 13 Supplement 7, 2020: Proceedings of the 7th iDASH Privacy and Security Workshop 2018*. The full contents of the supplement are available online at https://bmcmedgenomics.biomedcentral.com/articles/supplements/volume-13-supplement-7.

Authors' contributions

DH designed the overall algorithms for this study. YH and DW transformed the algorithms into homomorphic encryption algorithms. AD and SW contributed to the implementation of our HE algorithms. JH contributed in writing the manuscript. DH takes final responsibility for the manuscript. All author(s) have read and approved the final manuscript.

Funding

This work was supported in part by the Institute for Information & Communications Technology Promotion (IITP) Grant through the Korean Government (MSIT), (Development and Library Implementation of Fully Homomorphic Machine Learning Algorithms supporting Neural Network Learning over Encrypted Data), under Grant 2020-0-00840, in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) (No. 2017R1A5A1015626), and in part by the LG Electronics (LGE) grant.

Availability of data and materials

The dataset was available to participants registered to iDASH 2018 competition.

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Published: 21 July 2020

References

1. Malik MB, Ghazi MA, Ali R. Privacy preserving data mining techniques: current scenario and future prospects. In: Third International Conference on Computer and Communication Technology (ICCCT). Allahabad: IEEE; 2012. p. 26–32.
2. IDASH 2018. http://www.humangenomeprivacy.org/2018/. Accessed 15 Jan 2019.
3. Cheon JH, Kim A, Kim M, Song Y. Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security. Cham: Springer; 2017. p. 409–37.
4. Cheon JH, Han K, Kim A, Kim M, Song Y. Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Cham: Springer; 2018. p. 360–84.
5. Han K, Kim A, Kim M, Song Y. Implementation of HEAAN. https://github.com/snucrypto/HEAAN. Accessed 12 July 2018.
6. Lauter K, López-Alt A, Naehrig M. Private computation on encrypted genomic data. In: International Conference on Cryptology and Information Security in Latin America. Cham: Springer; 2014. p. 3–27.
7. Wang S, Zhang Y, Dai W, Lauter K, Kim M, Tang Y, Xiong H, Jiang X. Healer: homomorphic computation of exact logistic regression for secure rare disease variants analysis in GWAS. Bioinformatics. 2015;32(2):211–8.
8. Kim A, Song Y, Kim M, Lee K, Cheon JH. Logistic regression model training based on the approximate homomorphic encryption. BMC Med Genet. 2018;11(4):83.
9. Chen H, Gilad-Bachrach R, Han K, Huang Z, Jalali A, Laine K, Lauter K. Logistic regression over encrypted data from fully homomorphic encryption. BMC Med Genet. 2018;11(4):81.
10. Crawford JL, Gentry C, Halevi S, Platt D, Shoup V. Doing real work with FHE: The case of logistic regression. In: Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. New York: ACM; 2018. p. 1–12.
11. Bonte C, Vercauteren F. Privacy-preserving logistic regression training. BMC Med Genet. 2018;11(4):86.
12. IDASH 2017. http://www.humangenomeprivacy.org/2017/. Accessed 15 Jan 2019.
13. Lu W, Yamada Y, Sakuma J. Efficient secure outsourcing of genome-wide association studies. In: 2015 IEEE Security and Privacy Workshops. USA: IEEE; 2015. p. 3–6.
14. Bonte C, Makri E, Ardeshirdavani A, Simm J, Moreau Y, Vercauteren F. Towards practical privacy-preserving genome-wide association study. BMC Bioinformatics. 2018;19(1):537.
15. Jagadeesh KA, Wu DJ, Birgmeier JA, Boneh D, Bejerano G. Deriving genomic diagnoses without revealing patient genomes. Science. 2017;357(6352):692–5.
16. Cho H, Wu DJ, Berger B. Secure genome-wide association analysis using multiparty computation. Nat Biotechnol. 2018;36(6):547.
17. Kamm L, Bogdanov D, Laur S, Vilo J. A new way to protect privacy in large-scale genome-wide association studies. Bioinformatics. 2013;29(7): 886–93.
18. Constable SD, Tang Y, Wang S, Jiang X, Chapin S. Privacy-preserving GWAS analysis on federated genomic datasets. BMC Med Inform Decis Making. 2015;15:2. BioMed Central.
19. Bogdanov D, Kamm L, Laur S, Sokk V. Implementation and evaluation of an algorithm for cryptographically private principal component analysis on genomic data. IEEE/ACM Trans Comput Biol Bioinforma. 2018;15(5): 1427–32.
20. Chen F, Wang S, Jiang X, Ding S, Lu Y, Kim J, Sahinalp SC, Shimizu C, Burns JC, Wright VJ, et al. Princess: Privacy-protecting rare disease international network collaboration via encryption through software guard extensions. Bioinformatics. 2016;33(6):871–8.
21. Anati I, Gueron S, Johnson S, Scarlata V. Innovative technology for cpu based attestation and sealing. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy vol. 13. New York: ACM; 2013.
22. Kim M, Song Y, Wang S, Xia Y, Jiang X. Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR Med Inform. 2018;6(2):e19.
23. Cheon JH, Kim D, Kim Y, Song Y. Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. IEEE Access. 2018;6:46938–48.
24. Cheon JH, Han K, Hong SM, Kim HJ, Kim J, Kim S, Seo H, Shim H, Song Y. Toward a secure drone system: Flying with real-time homomorphic authenticated encryption. IEEE Access. 2018;6:24325–39. https://doi.org/10.1109/ACCESS.2018.2819189.
25. Albrecht MR, Player R, Scott S. On the concrete hardness of learning with errors. J Math Cryptol. 2015;9(3):169–203.
26. Albrecht MR. A sage module for estimating the concrete security of learning with errors instances. https://bitbucket.org/malb/lwe-estimator. Accessed 15 July 2018.
27. Sikorska K, Lesaffre E, Groenen PF, Eilers PH. Gwas on your notebook: fast semi-parallel linear and logistic regression for genome-wide association studies. BMC Bioinformatics. 2013;14(1):166.
28. Longford NT. A fast scoring algorithm for maximum likelihood estimation in unbalanced mixed models with nested random effects. Biometrika. 1987;74(4):817–27.
29. Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. 2016.
30. Juvekar C, Vaikuntanathan V, Chandrakasan A. Gazelle: A low latency framework for secure neural network inference. In: 27th USENIX Security Symposium (USENIX Security 18). Berkeley: USENIX Association; 2018.
31. Goldschmidt RE. Applications of division by convergence. PhD thesis, Massachusetts Institute of Technology. 1964.
32. Markstein P. Software division and square root using goldschmidt's algorithms. Proc 6th Conf Real Numbers Comput (RNC'6). 2004;123: 146–57.
33. Chinchor N. Muc-4 evaluation metrics. In: Proceedings of the 4th Conference on Message Understanding. USA: Association for Computational Linguistics; 1992. p. 22–9.

Publisher's Note