cryptography

Article

Implementing Privacy-Preserving Genotype Analysis with Consideration for Population Stratification

Andre Ostrak [1,2], Jaak Randmets [1], Ville Sokk [1], Sven Laur [3] and Liina Kamm [1,*]

1 Cybernetica AS, 12618 Tallinn, Estonia; andre.ostrak@cyber.ee (A.O.); jaak.randmets@cyber.ee (J.R.); ville.sokk@cyber.ee (V.S.)
2 Institute of Mathematics and Statistics, University of Tartu, 51009 Tartu, Estonia
3 Institute of Computer Science, University of Tartu, 51009 Tartu, Estonia; sven.laur@ut.ee
* Correspondence: liina.kamm@cyber.ee

Abstract: In bioinformatics, genome-wide association studies (GWAS) are used to detect associations between single-nucleotide polymorphisms (SNPs) and phenotypic traits such as diseases. Significant differences in SNP counts between case and control groups can signal association between variants and phenotypic traits. Most traits are affected by multiple genetic locations. To detect these subtle associations, bioinformaticians need access to more heterogeneous data. Regulatory restrictions in cross-border health data exchange have created a surge in research on privacy-preserving solutions, including secure computing techniques. However, in studies of such scale, one must account for population stratification, as under- and over-representation of sub-populations can lead to spurious associations. We improve on the state of the art of privacy-preserving GWAS methods by showing how to adapt principal component analysis (PCA) with stratification control (EIGENSTRAT), FastPCA, EMMAX and the genomic control algorithm for secure computing. We implement these methods using secure computing techniques—secure multi-party computation (MPC) and trusted execution environments (TEE). Our algorithms are the most complex ones at this scale implemented with MPC. We present performance benchmarks and a security and feasibility trade-off discussion for both techniques.

Keywords: privacy-preserving GWAS; secure multi-party computation; privacy-preserving statistics; trusted execution environments

check for updates

Citation: Ostrak, A.; Randmets, J.; Sokk, V.; Laur, S.; Kamm, L. Implementing Privacy-Preserving Genotype Analysis with Consideration for Population Stratification. *Cryptography* **2021**, *5*, 21. https://doi.org/10.3390/cryptography5030021

Academic Editor: Josef Pieprzyk

Received: 29 April 2021
Accepted: 18 August 2021
Published: 20 August 2021

1. Introduction

Genome-wide association studies (GWAS) split the cohort of individuals into two or more groups based on a phenotypic trait of interest, e.g., on the level of hemoglobin, on the existence or non-existence of a disease. These groups are compared to each other in the framework of case–control studies to find in the DNA sequence single-nucleotide polymorphisms (SNPs) that are significantly overrepresented in one group. GWAS have been performed extensively to date, and various concerns have been found. At least two of these problems—the existence of polygenic phenotypes, and population stratification—can be alleviated with the use of more heterogenous databases with larger volumes of data.

Polygenic phenotypes are traits that are affected by multiple genetic locations. Only a small subset of these locations are known, and they explain only a small amount of variance. To study them further, bioinformaticians need access to more heterogeneous data in order to detect subtle associations.

Population stratification is most often caused by geographic isolation of subpopulations over several generations [1]. If the case and control groups are compiled carelessly, genetic differentiation caused by population stratification can confound associations between genotype and trait. The resulting false positive or negative associations may, in fact, result from differences in local ancestry and be independent from the phenotypic trait under investigation [2]. An example of allele frequencies differing because of ancestry was shown in a case–control study [3]. In this study, a European American cohort was investigated. They found that a SNP in the lactase gene (LCT) gave a strong statistically significant association ($p < 10^{-6}$) with height. Both height and the LCT gene have wide variations across populations in Europe. This spurious association was reduced when individuals were rematched on the basis of European ancestry.

While collaboration among biobanks is improving, the construction of datasets that represent continent-wide or worldwide populations pose significant privacy risks. To alleviate these risks, privacy-preserving methods with cryptographic guarantees have been identified as the potential solution. Moreover, the European Data Protection Board (EDPB) has acknowledged certain secure computing technologies (split or multi-party processing) as an effective technical measure for protecting personal data transfers across borders [4]. This comes in light of the recent Court of Justice of the European Union judgment in the Schrems II case. Furthermore, the European Data Protection Supervisor has noted the role of using technologies that enable computation over encrypted data in the context of creating the European Health Data Space [5]. Secure multi-party computation (MPC) is one such privacy enhancing technique that can provide cryptographic guarantees. Two organisational models for GWAS-on-MPC were shown already in [6]. A number of implementations have been proposed since, including [7–13]. Components of GWAS have also been implemented using fully homomorphic encryption [14,15] and Intel® Software Guard eXtensions (SGX) trusted execution environments [16–19].

However, even in big heterogenous datasets, population stratification or over-representation of sub-populations can lead to spurious associations. To counteract these effects, bioinformaticians use estimation methods that propose parameters that represent the ancestry of the cohort. The number of these associated SNPs can greatly vary based on how different or similar the populations are. Principal component analysis (PCA) uses eigenvector decomposition to infer population stratification. The results can be used to visualise genetic relationships between variables alongside reference populations helping identify population stratification. Alternatively, the results can be used to adjust the genotypes and phenotypes for stratification as in [20]. Stand-alone PCA for GWAS using MPC has been demonstrated in [8,10,21]. A survey paper of privacy-preserving techniques for bioinformatics was published in 2017 [22].

In this paper, we show how the state-of-the-art GWAS methods that account for population stratification can be made to be privacy-preserving. That is, we present privacy-preserving versions of the following four algorithms: EIGENSTRAT [20], FastPCA [23], EMMAX [24] and genomic control [25]. We redesign the existing widely-used algorithms to use two different privacy enhancing technologies (PETs)—secure multi-party computation and trusted execution environments. The algorithms can be used on multiple protocol backends in various security models. We compare the solutions, implement the algorithms and present benchmark results. For implementation, we use the Sharemind MPC [26] and Sharemind HI platforms. Sharemind HI uses the Intel® Software Guard eXtensions (SGX) technology to create trusted execution environments and, thus, can be run securely in single server setting while Sharemind MPC requires a more complex setup.

2. Materials and Methods

2.1. Genome-Wide Association Studies

To conduct genome-wide association studies (GWAS), the cohort of individuals is usually split into two (or more) groups—cases and controls—based on a phenotypic trait of interest, e.g., the colour of the eyes. Next, the genotype data of the groups are studied to detect DNA sequence single-nucleotide polymorphisms (SNPs) that are significantly overrepresented in one group. Although all four nucleotides (A, C, G, and T) can be located at a SNP site, usually, only two alleles A and B are considered. The first (A) corresponds to the reference sequence, and the second (B) represents potential mutations. SNP data are stored as pairs of chromosomes (AA, AB, BB, and NN if the measurement could not be completed for some reason) in text format. It is also common to use binary encoding for

SNPs, where zero denotes the dominant nucleotide in the specific DNA location and one denotes the remaining three alternatives. As the same genetic information is represented in two chromosomes and common genotyping techniques measure these simultaneously, the resulting measurement outcome can be encoded as $\{\perp, 0, 1, 2\}$, where $\perp$ encodes measurement failures.

In GWAS, genotype data are usually represented as an $n \times m$ matrix $\mathbf{X}$ with values $\{0, 1, 2\}$, where the rows correspond to individuals and columns to SNP locations. Phenotype data are given as a $\{0, 1\}$ vector $\mathbf{y}$ of length n, where 1 denotes the case group (the existence of the trait(s) in question) and 0 denotes the control group (the lack of the trait(s) in question). The measurement failures are denoted by the $n \times m$ mask matrix $\mathbf{M}$ with values $\{0, 1\}$ that indicate whether the corresponding SNP is available or not (0 represents the missing SNP and 1 the available SNP). Table 1 gives an example what this dataset looks like.

Table 1. Example genotype dataset and phenotype vector for GWAS.

Donor ID	SNP_1	SNP_2	...	SNP_m	**Phenotype Vector y**
D_1	0	1	...	0	0
D_2	1	0	...	0	1
...	...	...	...	...	...
D_n	2	0	...	2	1

2.2. *Algorithms for GWAS That Account for Population Stratification*

To assess the effect of SNP i on a phenotypic trait in an additive model, the allelic data of the marker can be presented in a table as described in Table 2. For a perfectly mixed population, the marker can be tested with the χ^2-test. However, population substructure and inbreeding can lead to an increased homozygosity, which induces extra-binomial variance, leading to spurious associations. Within a genotype, this increased variance can be accounted for with the Cochran–Armitage trend test, where each SNP i can be tested for whether it satisfies the following assumption:

$$Y_i^2 = \frac{N_i(N_i(r_{i1} + 2r_{i2}) - R_i(n_{i1} + n_{i2}))^2}{R_i(N_i - R_i)(N_i(n_{i1} + 4n_{i2}) - (n_{i1} + 2n_{i2})^2)} \sim \chi^2 \ .$$

However, the trend test does not take into account the increased allelic variance between genotypes caused by population substructures and hidden relatedness. We selected four algorithms, which are used to account for population stratification, and implemented them using secure computing techniques.

Table 2. Genotype distribution for the Cochran-Armitage trend test.

Group	AA	AB	BB	**Total**
Case	r_0	r_1	r_2	R
Control	s_0	s_1	s_2	S
Total	n_0	n_1	n_2	N

Genomic control. To correct for the effect of population stratification, this algorithm [25] estimates the variance inflation factor, which measures the increase in allelic variance across individuals within the samples. Genomic control assumes that this factor is a constant for all markers, and it can therefore be estimated using the Cochran–Armitage test statistics of all markers. To lower the rate of false positives caused by cryptic relatedness and population stratification, the test statistics are divided by the computed estimate. The resulting statistics are approximately distributed χ^2 with one degree of freedom under the null hypothesis.

While powerful, it has its limitations. Mainly, if the markers are not uniformly differentiated across ancestral populations, the estimation of the inflation factor can be too small, and therefore insufficient, for some markers, leading to a number of false positive associations. For other markers, the inflation factor can be too large, thereby becoming superfluous.

Principal component analysis. It has been shown that for samples of different subpopulations, the first principal components of the genotype data describe the differences in ancestry [20]. Therefore, it is possible to correct for stratification by adjusting the genotype and phenotype data using the principal components as covariates, as is done in the EIGENSTRAT algorithm [20]. For hypothesis testing, a generalisation of the Cochran–Armitage trend test can be used on the adjusted genotypes and phenotypes. Unlike the genomic control algorithm, this approach does not adjust all markers uniformly. Therefore, it can better capture the differences in allele frequencies caused by different ancestrial populations.

The main drawback of principal component analysis is that it is often unclear how many and which principal components capture the population substructure and what other top principal components capture in the sample structure. Usually up to 10 principal components are used to adjust for population stratification.

FastPCA [23] is a variation of principal component analysis. It uses recent advances in random matrix theory to reduce the computational effort in approximating the first principal components, which are simply the top eigenvectors of the kinship matrix. We chose FastPCA because computing eigenvectors in the privacy-preserving environment is very time consuming, as are all operations that deal with floating-point computations, and we do not need to compute exact eigenvectors.

EMMAX. An even more elaborate way is to use linear mixed models to correct for population stratification with each SNP as a fixed effect [24]. The relatedness of individuals in the sample are captured by the variance components. To assess the effect of the SNP to the phenotype, we first estimate these variance components, after which we can test the significance of the fixed effect by using a t-test.

The EMMAX algorithm is computationally more demanding than the previous algorithms. However, it is often statistically more powerful because it is able to better capture and correct for cryptic relatedness, especially for smaller sample substructures.

2.3. Privacy-Preserving Computation and GWAS

2.3.1. Deployment Models for Privacy-Preserving GWAS

In a typical collaborative genome-wide association study setting, two or more biobanks or service providers run collaborative analysis on data they have collected from their donors. Due to regulatory requirements or personal preferences, the processing of donor data should be minimised so that data leaks can be kept to a minimum. However, donors may still be willing to participate in research if a privacy-preserving solution is available.

Each biobank holds genotype data and medical diagnoses of a number of individuals. The analyst wants to run the genome-wide association study on the combined data of the gene banks. Since a person's genotype data are sensitive, the banks cannot reveal their data to each other. The analyst must not be able to determine any information about the individuals in the study by performing the analysis, and they should only receive the names of the significant SNPs. For a more detailed analysis of the usage models of MPC in GWAS, refer to [6].

Either the donors or the biobanks act as the input parties, who provide protected data for the computation. The biobanks, universities or dedicated service providers act as computing parties who host and run the secure computing system. The system is used by researchers in, e.g., universities or pharmaceutical companies who want to study a specific disease. They are the result parties in the system.

2.3.2. Cryptographic Secure Multi-Party Computation

In secure multi-party computation (MPC), two or more parties compute a function without seeing any of the private input values of the other parties. Most often secret sharing, garbled circuits or homomorphic encryption are used to enable MPC [27]. MPC technology has reached a level of maturity required to enable real-world implementations [28]. Data processing applications are typically implemented using one of the programmable MPC frameworks [29].

In this paper, we prototype our implementation using the SHAREMIND MPC platform [26]. SHAREMIND MPC is a distributed platform for privacy-preserving data processing supporting multiple MPC protocol sets, which use secret sharing as the secure storage method. Each value that needs to be protected is shared by the input parties according to the secret sharing algorithm. The resulting random shares are distributed between multiple computing parties.

Each computing party has a copy of the algorithms and runs them using MPC protocols that convert secret-shared inputs into secret-shared outputs, without recovering the private values on any computing party. Such MPC systems provide end-to-end security for data processing if a sufficient number of computing parties follow the protocol. If one or more do not, it may cause the computation to reach the wrong result or even breach privacy. The particular protocol set determines the exact failure mode. Efficient protocol sets are available for two and three computing parties, and adding more parties increases the communication complexity, a common bottleneck in MPC based on secret sharing.

2.3.3. Trusted Execution Environments

A trusted execution environment (TEE) is a secure area of a processor. It ensures the confidentiality and integrity of the the data that are loaded into this area. It is isolated from the rest of the processor, and the computations cannot be monitored. In a standard deployment, data are first sent to the TEE. Next, either computations are run immediately or data can be sealed and stored outside the TEE (kept on the disk in an encrypted form). For computing, the data are loaded into the TEE and decrypted there. After the computations have been completed, results are either encrypted and stored on the disk, or published. The host will not have access to the encryption keys for the data nor be able see decrypted data in any other way.

The Intel® Software Guard eXtensions (SGX) has emerged as a popular way to create trusted execution environments. SGX provides features such as enclaves, attestation and data sealing to protect data and allow for remote audit of processing and privacy policy enforcement. In SGX, enclaves are protected memory areas that protect the confidentiality and integrity of data even if there is malware in the system. Attestation helps the system prove to external parties that it is running the trusted software and the correct version of the application. Data sealing (encryption) is used to store data outside the enclave while protecting its confidentiality and integrity. Only the specific enclave knows the encryption key and can decrypt the data.

Trusted execution environments do not require distributed storage; thus they can be used to build a system where there is just a single computing party. However, that computing party has a different trust model than computing parties in MPC. Notably, one has to trust the provider of the hardware that is used to create the trusted execution environments and that it has been implemented correctly. Second, applications of TEEs need to account for side channel attacks that observe execution time, power usage or the electromagnetic radiation of the hardware while the privacy-preserving task is running. These can be used to recover the encrypted data in the enclave. At the same time, TEEs can perform privacy-preserving operations significantly faster than MPC, and thus the trade-off between their secure models requires study. See [30] for a recent review of SGX attacks and their mitigations.

SHAREMIND HI (HI standing for hardware isolation) is a privacy-preserving data processing platform that uses SGX enclaves, sealing, and attestation to support data analysis processes between identified participants and provides end-to-end privacy for the protected data for applications just as SHAREMIND MPC does.

2.3.4. Adapting Algorithms for Secure Computing

The privacy-preserving algorithms that we design in this paper minimise data leakage. Our design process follows the security goals for privacy-by-design statistical analysis algorithms as defined by [31] (The authors of [31] also note the importance of query restrictions that prevent new, possibly insecure, algorithms from being executed on a secure computing platform before whitelisting. However, this does not affect the design of algorithms and only affects the choice of the secure computing platform the algorithms will run on).

1. **Cryptographic security.** During computation, no computing party learns any private input or intermediate value computed by the function unless the value is explicitly published. Private values are also not leaked through changes in the running time of the algorithm. We achieve this by designing algorithms that process all private values either by using MPC and TEEs or by making the algorithm running time as independent of the inputs as possible. This means designing the algorithms as straight-line programs that have no branching based on private values or where each branching decision is analysed with regard to what it could leak in the running time.
2. **Source privacy.** An algorithm is source-private if all its outputs and all intermediate values do not depend on the order of inputs. The GWAS algorithms in this paper are designed to avoid leaking information based on the order of donors' data in the inputs. The order of SNPs in the inputs will affect the SNPs in the output, but this is by design as the related metadata are not private data.
3. **Output privacy.** An algorithm is output-private if the results it declassifies do not leak the private inputs. The GWAS algorithms in this paper can be used either as parts of larger processing workflows, and in this case they do not declassify outputs. However, the results of the algorithms are statistical significances per SNP and contain no personal data given a sufficient amount of inputs.

The algorithms also need adaptation for performance. In secure multi-party computation based on secret sharing, servers need to communicate with each other to run the shared computations. It has been shown that redesigning algorithms with maximum parallelisation helps use the communication channel more efficiently, reducing the round count and improving speed. Thus, the algorithms in this paper are adapted to use single-instruction, multiple-data (SIMD) vector and matrix operations to the maximum reasonable extent.

The algorithms are designed for secure computing frameworks supporting integer, fixed and floating-point arithmetic. In this paper, we implement the algorithms using the SHAREMIND MPC and SHAREMIND HI platforms as they support a comparable application model on different secure computing technologies. This allows us to compare the techniques. However, the algorithms would work on any secure computing technique, including fully homomorphic encryption or zero knowledge.

In the following algorithm representations, we use the notation $[\![x]\!]$ to denote a value x that is protected by the selected secure computing system. For example, in an MPC runtime based on secret sharing, $[\![x]\!]$ means that x has been secretly shared among multiple parties so that no party can recover it. For trusted execution environments, $[\![x]\!]$ means that x is only processed within the enclave. Similarly, $[\![\mathbf{x}]\!]$, $[\![\mathbf{X}]\!]$ denote protected data structures (vector $\mathbf{x}$ and matrix $\mathbf{X}$, respectively). For a matrix $\mathbf{X}$ and indexes i, j, k, l, we denote by $\mathbf{X}[i : j, k : l]$ the submatrix of $\mathbf{X}$ consisting of elements from rows $i, \dots, j-1$ and columns $k, \dots, l-1$. A missing index means that we take all the elements up until or starting from an index.

The operator $*$ denotes element-wise multiplication between two matrices or multiplication by a scalar and is used to distinguish this operation from matrix multiplication. The operator/denotes element-wise division by scalar, vector or matrix, as necessary. The functions **rowSums** and **colSums** compute the sum of each row or column of a matrix, respectively. The output is a vector.

The function **reshape** takes as input a vector and two integers n and m and reshapes the vector row-wise into an $n \times m$ matrix. The function **flatten** flattens a matrix row-wise into a vector. The function **choose** takes as inputs a Boolean vector $[\![\mathbf{m}]\!]$ and two vectors

$[\![\mathbf{x}]\!]$ $[\![\mathbf{y}]\!]$ of the same type and chooses elements point-wise from either $[\![\mathbf{x}]\!]$ or $[\![\mathbf{y}]\!]$ based on the values in vector $[\![\mathbf{m}]\!]$. The operator **cut** takes as arguments a Boolean vector $[\![\mathbf{m}]\!]$ and a same size vector $[\![\mathbf{x}]\!]$ of any type and discards the elements point-wise from $[\![\mathbf{x}]\!]$ if the corresponding value in $[\![\mathbf{m}]\!]$ is false. While the size of the output vector can leak information, since the size of the output vector is equal to the number of true values in $[\![\mathbf{m}]\!]$, in our implementations, we make sure that the function always outputs a vector with a single value. This guarantees that no information is leaked.

3. Results

In this section, we describe the privacy-preserving versions of the algorithms and give the benchmark results. We first describe and discuss EIGENSTRAT and FastPCA, and then we go to EMMAX and finally we describe the genomic control algorithm. The EIGENSTRAT and EMMAX algorithms use the privacy-preserving GS-PCA algorithm from [8]. GS-PCA is highly parallelisable and therefore especially suits the secure multi-party setting. Our privacy-preserving genomic control algorithm uses the privacy-preserving version of quicksort from [32].

The nature of the algorithm adaptation was twofold: firstly, we made the modifications that were needed for the algorithm to work in the privacy-preserving setting, and secondly, we added implementation details for SHAREMIND, the MPC framework we used. To lower the computing time while maintaining accuracy, our MPC implementations use a mix of floating-point and fixed-point arithmetic. The conversion was only done in cases where the accuracy loss was at acceptable levels.

It is important to note that the algorithms can be viewed separately from their implementations, as the algorithms can be implemented for any MPC platform that has the necessary fixed-point and floating-point arithmetic available. The fixed-point and floating-point conversions that are discussed in these subsections can be beneficial for implementations on other platforms as well, and, therefore, we have added these directly into the algorithm description.

3.1. Privacy-Preserving EIGENSTRAT and FastPCA

Algorithm 1 describes the privacy-preserving version of EIGENSTRAT [20]. As input it receives the genotype matrix $[\![\mathbf{X}]\!]$, the corresponding matrix of mask vectors $[\![\mathbf{M}]\!]$, and the phenotype vector $[\![\mathbf{y}]\!]$. It also receives the number of components k and the number of GS-PCA iterations J. According to [20], k is usually 10 but can also be 1, 2 or 5. In our benchmark tests, we set $k = 2$. The GS-PCA algorithm requires the number of iterations J as an input. Using a public predetermined number of iterations avoids side channel attacks based on the number of iterations. In our benchmarks, we set J to 15 as this gives us sufficient precision.

We start by computing the vector $[\![\boldsymbol{\mu}]\!]$ of the means of the columns and use this to normalise the columns. In the MPC implementation, we use floating-point arithmetic to compute the normalised genotype matrix $[\![\mathbf{Z}]\!]$, and we convert this matrix to fixed-point values before finding the scaled kinship matrix $[\![\mathbf{K}]\!] \leftarrow [\![\mathbf{Z}]\!][\![\mathbf{Z}]\!]^T$, which is $n-1$ times the covariance matrix. We do not perform the division with $n-1$, as this is an expensive operation, and it does not influence the properties of eigenvalues that we need for our computations.

Using the privacy-preserving GS-PCA algorithm, we find the vector $[\![\boldsymbol{\lambda}]\!]$ of the k largest eigenvalues and the $n \times k$ matrix $[\![\mathbf{U}]\!]$ of corresponding eigenvectors of matrix $[\![\mathbf{K}]\!]$. In this algorithm, we do not need the vector $[\![\boldsymbol{\lambda}]\!]$ of eigenvalues, so we simply drop this result. Our MPC implementation of the GS-PCA algorithm also uses fixed-point arithmetic. To adjust for stratification, the entries are converted back to floating-point values as this keeps the accuracy loss to a minimum. Using the eigenvectors from $[\![\mathbf{U}]\!]$, we iteratively adjust the genotype matrix and phenotype vector and then compute the trend statistics using the adjusted values. No values are declassified during the execution.

The biggest issue with this algorithm in the privacy-preserving setting is that finding $[\![\mathbf{ZZ}^T]\!]$ requires a lot of computational effort. The computation time of the GS-PCA algorithm also increases quadratically as the number of rows in $[\![\mathbf{Z}]\!]$ (the size of the sample) increases. To deal with this, we can instead use FastPCA [23], which makes use of random matrix theory.

Algorithm 1: Privacy-preserving EIGENSTRAT

```
Input: Genotype matrix $[\![\mathbf{X}]\!]$ (private), mask matrix $[\![\mathbf{M}]\!]$ (private), phenotype vector $[\![\mathbf{y}]\!]$ (private), number of
       components $k$ (public), number of GS-PCA iterations $J$ (public)
Output: Private vector of statistics $[\![\mathbf{stat}]\!]$ for the $\chi^2$-test
1  $[\![\mathbf{sumM}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{M}]\!])$
2  $[\![\mu]\!] \leftarrow \mathbf{colSums}([\![\mathbf{X}]\!])/[\![\mathbf{sumM}]\!]$
3  $[\![\mathbf{p}]\!] \leftarrow \mathbf{colSums}(1 + [\![\mathbf{X}]\!])/(2 + 2[\![\mathbf{sumM}]\!])$
4  $[\![\mathbf{div}]\!] \leftarrow \mathbf{sqrt}([\![\mathbf{p}]\!](1 - [\![\mathbf{p}]\!])$
5  $[\![\mu\mathbf{Mat}]\!] \leftarrow [\![\mu]\!]$ // $\mu Mat$ is an $n \times m$ matrix, each row is equal to $\mu$.
6  $[\![\mathbf{divMat}]\!] \leftarrow [\![\mathbf{div}]\!]$ // $\mathbf{divMat}$ is an $n \times m$ matrix, each row is equal to $\mathbf{div}$.
7  $[\![\mathbf{Z}]\!] \leftarrow ([\![\mathbf{X}]\!] - [\![\mathbf{M}]\!] * [\![\mu\mathbf{Mat}]\!])/[\![\mathbf{divMat}]\!]$
8  $[\![\mathbf{K}]\!] \leftarrow [\![\mathbf{Z}]\!][\![\mathbf{Z}]\!]^T$
9  $[\![\lambda]\!], [\![\mathbf{U}]\!] \leftarrow \mathbf{GS\text{-}PCA}([\![\mathbf{K}]\!], k, J)$
10 $[\![\mathbf{X_{adj}}]\!] \leftarrow [\![\mathbf{X}]\!]$
11 $[\![\mathbf{y_{adj}}]\!] \leftarrow [\![\mathbf{y}]\!]$
12 for $j = 0, \ldots, k-1$ do
13     $[\![\mathbf{U_j}]\!] \leftarrow [\![\mathbf{U}[j,:]]\!]$
14     $[\![\mathbf{U_j^2}]\!] \leftarrow [\![\mathbf{U_j}]\!] * [\![\mathbf{U_j}]\!]$
15     $[\![\mathbf{U_jMat}]\!] \leftarrow [\![\mathbf{U_j}]\!]$ // $\mathbf{U_jMat}$ is an $n \times m$ matrix, each row is equal to $\mathbf{U_j}$.
16     $[\![\mathbf{U_j^2Mat}]\!] \leftarrow [\![\mathbf{U_j^2}]\!]$ // $\mathbf{U_j^2Mat}$ is an $n \times m$ matrix, each row is equal to $\mathbf{U_j^2}$.
17     $[\![\mathbf{U_j^2M}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{U_j^2Mat}]\!] * [\![\mathbf{M}]\!])$
18     $[\![\mathbf{U_jM}]\!] \leftarrow [\![\mathbf{U_jMat}]\!] * [\![\mathbf{M}]\!]$
19     $[\![\gamma]\!] \leftarrow \mathbf{colSums}([\![\mathbf{U_jM}]\!] * [\![\mathbf{X_{adj}}]\!])/[\![\mathbf{U_j^2M}]\!]$
20     $[\![\gamma\mathbf{Mat}]\!] \leftarrow [\![\gamma]\!]$ // $\gamma Mat$ is an $n \times m$ matrix, each row is equal to $\gamma$
21     $[\![\mathbf{X_{adj}}]\!] \leftarrow [\![\mathbf{X_{adj}}]\!] - [\![\gamma\mathbf{Mat}]\!] * [\![\mathbf{U_jM}]\!]$
22     $[\![\gamma_y]\!] \leftarrow \mathbf{sum}([\![\mathbf{U_j}]\!] * [\![\mathbf{y_{adj}}]\!])$
23     $[\![\mathbf{y_{adj}}]\!] \leftarrow [\![\mathbf{y_{adj}}]\!] - [\![\gamma_y]\!] * [\![\mathbf{U_j}]\!]$
24 end
25 $[\![\mathbf{y_{adj}Mat}]\!] \leftarrow [\![\mathbf{y_{adj}}]\!]$ // $\mathbf{y_{adj}Mat}$ is an $n \times m$ matrix, each row is equal to $\mathbf{y_{adj}}$
26 $[\![\mathbf{sumXy}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{X_{adj}}]\!] * [\![\mathbf{y_{adj}Mat}]\!]) * [\![\mathbf{sumM}]\!]$
27 $[\![\mathbf{sumX}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{X_{adj}}]\!])$
28 $[\![\mathbf{sumy}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{y_{adj}Mat}]\!])$
29 $[\![\mathbf{varX}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{X_{adj}}]\!] * [\![\mathbf{X_{adj}}]\!] * [\![\mathbf{M}]\!]) * [\![\mathbf{sumM}]\!] - [\![\mathbf{sumX}]\!] * [\![\mathbf{sumX}]\!]$
30 $[\![\mathbf{vary}]\!] \leftarrow \mathbf{colSums}([\![\mathbf{y_{adj}Mat}]\!] * [\![\mathbf{y_{adj}Mat}]\!] * [\![\mathbf{M}]\!]) * [\![\mathbf{sumM}]\!] - [\![\mathbf{sumy}]\!] * [\![\mathbf{sumy}]\!]$
31 return $[\![\mathbf{stat}]\!] \leftarrow ([\![\mathbf{sumM}]\!] - k - 1) * ([\![\mathbf{sumXy}]\!] - [\![\mathbf{sumX}]\!] * [\![\mathbf{sumy}]\!])^2/([\![\mathbf{varX}]\!] * [\![\mathbf{vary}]\!])$
```

Algorithm 2 describes the privacy-preserving version of the FastPCA randomised algorithm for finding the top k left singular vectors [23]. As input, it receives the normalised genotype matrix $[\![\mathbf{Z}]\!]$, the number of components k, the oversampling parameter p and the number of iterations J. We estimate that $J = 12$ is enough to ensure accuracy.

The privacy-preserving implementation is fairly straightforward. We use the Marsaglia polar method to generate the entries for the random matrix $[\![\mathbf{Gi}]\!]$. To speed up the process of finding $[\![\mathbf{U_k}]\!]$, we used fixed-point not floating-point values. The biggest difference in fixed and floating-point computation comes from computing $[\![\mathbf{G_i}(\mathbf{ZZ^T})^\mathbf{J}\mathbf{Z}]\!]$. While this process is faster when done with fixed-point values, it causes accuracy loss in some cases.

It is important to keep in mind that when using fixed-point values, we can get overflows during the iterative matrix multiplication. To prevent this, we divide the product matrix $[\![\mathbf{G_i}(\mathbf{ZZ}^T)^i]\!]$ with a suitable constant after each iterative step.

For QR factorisation and eigendecomposition, we used the Householder QR algorithm and the symmetric QR algorithm [33], respectively. Having found the k top principal components, we can use them to adjust our genotype and phenotype vectors like in EIGENSTRAT. Again, no values are declassified during the execution.

Algorithm 2: Privacy-preserving FastPCA randomised algorithm for top k left singular vectors

```
Input: Normalised genotype matrix $[\![\mathbf{Z}]\!]$ (private), number of components $k$ (public), oversampling parameter $p$
       (public), number of iterations $J$ (public)
Output: Matrix $[\![\mathbf{U_k}]\!]$ of $k$ top eigenvectors of $\mathbf{ZZ}^T$ (private)
1  $n \leftarrow$ shape($[\![\mathbf{Z}]\!]$)[0] // sample size
2  $[\![\mathbf{Gi}]\!] \leftarrow$ normal$(k+p, n)$ // generating a $(k+p) \times n$ matrix whose entries are from the standard Gaussian
     distribution, uses the Marsaglia polar method.
3  for $i = 0, \ldots, J-1$ do
4      $[\![\mathbf{Hi}]\!] \leftarrow [\![\mathbf{Gi}]\!][\![\mathbf{Z}]\!]$
5      $[\![\mathbf{Gi}]\!] \leftarrow [\![\mathbf{Hi}]\!][\![\mathbf{Z}]\!]^T$
6      $[\![M]\!] \leftarrow$ max(flatten($[\![\mathbf{Gi}]\!]$))
7      $[\![\mathbf{Gi}]\!] \leftarrow [\![\mathbf{Gi}]\!] / [\![M]\!]$
8  end
9  $[\![\mathbf{Hi}]\!] \leftarrow [\![\mathbf{Gi}]\!][\![\mathbf{Z}]\!]$
10 $[\![\mathbf{Q}]\!] \leftarrow$ QR($[\![\mathbf{Hi}]\!]^T$) // Householder QR factorisation
11 $[\![\mathbf{T}]\!] \leftarrow [\![\mathbf{Z}]\!][\![\mathbf{Q}]\!]$
12 $[\![\mathbf{TT}]\!] \leftarrow [\![\mathbf{T}]\!]^T[\![\mathbf{T}]\!]$
13 $[\![\lambda]\!], [\![\tilde{\mathbf{W}}]\!] \leftarrow$ eigendecomposition($[\![\mathbf{TT}]\!]$) // $\lambda$ is the $(k+p)$ vector of eigenvalues and $\tilde{\mathbf{W}}$ the $(k+p) \times (k+p)$
     matrix of eigenvectors
14 $[\![\lambda\mathbf{inv}]\!] \leftarrow 1/$sqrt($[\![\lambda[:k]]\!]$)
15 $[\![\lambda\mathbf{invMat}]\!] \leftarrow [\![\lambda\mathbf{inv}]\!]$ // $\lambda\mathbf{invMat}$ is $(k+p) \times k$ matrix, each row is equal to $\lambda\mathbf{inv}$
16 $[\![\mathbf{T\tilde{W}}]\!] \leftarrow [\![\mathbf{T}]\!][\![\tilde{\mathbf{W}}[:,:k]]\!]$
17 return $[\![\mathbf{U_k}]\!] \leftarrow [\![\mathbf{T\tilde{W}}]\!] * [\![\lambda\mathbf{invMat}]\!]$
```

3.2. *Privacy-Preserving EMMAX*

Algorithm 3 describes the privacy-preserving version of EMMAX [24]. Although the algorithm can generally be used for the analysis of quantitative phenotypic traits, we implemented it with case–control studies in mind. Therefore, as input, the algorithm takes a similar genotype matrix $[\![\mathbf{X}]\!]$ as EIGENSTRAT, the mask matrix $[\![\mathbf{M}]\!]$ indicating the missing values, the phenotype vector $[\![\mathbf{y}]\!]$ indicating the inclusion into the case group, and the number of GS-PCA iterations J.

We use a mix of fixed-point and floating-point arithmetic in our MPC implementation of the EMMAX algorithm. We start by normalising the columns of $[\![\mathbf{X}]\!]$. As with the privacy-preserving EIGENSTRAT, we use floating-point arithmetic to compute the normalised genotype matrix $[\![\mathbf{Z}]\!]$ and convert the entries to fixed-point values to compute the the covariance matrix $[\![\mathbf{K}]\!] \leftarrow [\![\mathbf{Z}]\!][\![\mathbf{Z^T}]\!]$. We then use the GS-PCA algorithm to find the eigendecomposition of $[\![\mathbf{K}]\!]$. The resulting eigenvectors and eignevalues are converted to floating-point values again. The rest of the computations in the MPC implementation use floating-point arithmetic. This keeps the accuracy loss to a minimum.

We denote by $[\![\xi]\!]$ the vector of eigenvalues and by $[\![\mathbf{U_F}]\!]$ the matrix of corresponding eigenvectors, by $[\![\lambda]\!]$ the vector of $n-1$ largest eigenvalues and by $[\![\mathbf{U_R}]\!]$ the matrix of corresponding eigenvectors. After finding the vector $[\![\eta]\!] = [\![\mathbf{U_R}]\!]^T[\![\mathbf{y}]\!]$, we create a $101 \times (n-1)$ matrix $\delta\mathbf{Mat}$, where every column has values ranging from 10^{-5} to 10^5. Creating $101 \times (n-1)$ matrices $[\![\eta\mathbf{2Mat}]\!]$ and $[\![\lambda\mathbf{Mat}]\!]$, where the columns are vectors $[\![\eta^2]\!]$ and $[\![\lambda]\!]$, respectively, allows us to compute the value of the restricted log-likelihood function

f_R at each δ in parallel. Amongst these, we choose the δ for which f_R obtains the greatest value, roughly approximating the maximum point of the restricted likelihood function. We improve upon this approximation by using the Newton–Rhapson method to obtain $\llbracket \delta_{max} \rrbracket$.

Algorithm 3: Privacy-preserving EMMAX

```
Input: Genotype matrix $\llbracket \mathbf{X} \rrbracket$ (private), mask matrix $\llbracket \mathbf{M} \rrbracket$ (private), phenotype vector $\llbracket \mathbf{y} \rrbracket$ (private), number of
       GS-PCA iterations $J$ (public)
Output: Vector $\mathbf{t}$ of statistics for the t-test
1  $\llbracket \mu \rrbracket \leftarrow \mathbf{colSums}(\llbracket \mathbf{X} \rrbracket)/\mathbf{colSums}(\llbracket \mathbf{M} \rrbracket)$
2  $\llbracket \mathbf{Z} \rrbracket \leftarrow (\llbracket \mathbf{X} \rrbracket - \llbracket \mathbf{M} \rrbracket * \llbracket \mu \rrbracket)/(\mathbf{sqrt}(\llbracket \mu \rrbracket/2(1 - \llbracket \mu \rrbracket/2)))$
3  $\llbracket \mathbf{K} \rrbracket \leftarrow \llbracket \mathbf{Z} \rrbracket \llbracket \mathbf{Z} \rrbracket^T$
4  $\llbracket \xi \rrbracket, \llbracket \mathbf{U_F} \rrbracket \leftarrow \mathbf{GS\text{-}PCA}(\llbracket \mathbf{K} \rrbracket, n, J)$
5  $\llbracket \lambda \rrbracket \leftarrow \llbracket \xi[: n-1] \rrbracket$
6  $\llbracket \mathbf{U_R} \rrbracket \leftarrow \llbracket \mathbf{U_F}[:, : n-1] \rrbracket$
7  $\llbracket \eta \rrbracket \leftarrow \llbracket \mathbf{U_R} \rrbracket^T \llbracket \mathbf{y} \rrbracket$
8  $\llbracket \eta\mathbf{2} \rrbracket \leftarrow \llbracket \eta \rrbracket * \llbracket \eta \rrbracket$
9  $\llbracket \eta\mathbf{2Mat} \rrbracket \leftarrow \llbracket \eta\mathbf{2} \rrbracket$ // $\boldsymbol{\eta}\mathbf{2Mat}$ is a $101 \times (n-1)$ matrix, each row is equal to $\boldsymbol{\eta}\mathbf{2}$
10 $\llbracket \lambda\mathbf{Mat} \rrbracket \leftarrow \llbracket \lambda \rrbracket$ // $\lambda\mathbf{Mat}$ is a $101 \times (n-1)$ matrix, each row is equal to $\boldsymbol{\lambda}$
11 $\delta \leftarrow (10^{-5}, \dots, 10^5)$
12 $\delta\mathbf{Mat} \leftarrow \delta$ // $\delta\mathbf{Mat}$ is a $101 \times (n-1)$ matrix, each column is equal to $\delta$
13 $\llbracket \mathbf{\Lambda p}\delta \rrbracket \leftarrow \llbracket \lambda\mathbf{Mat} \rrbracket + \delta\mathbf{Mat}$
14 $\llbracket \mathbf{f_{pot}} \rrbracket \leftarrow -(n-1)\mathbf{ln}(\mathbf{colSums}(\llbracket \eta\mathbf{2Mat} \rrbracket/\llbracket \mathbf{\Lambda p}\delta \rrbracket)) - \mathbf{colSums}(\mathbf{ln}(\llbracket \mathbf{\Lambda p}\delta \rrbracket))$
15 $\llbracket \mathbf{f_{pot}} \rrbracket \leftarrow \llbracket \mathbf{f_{pot}} \rrbracket + \delta * (\mathbf{max}(\llbracket \mathbf{f_{pot}} \rrbracket) == \llbracket \mathbf{f_{pot}} \rrbracket)$ // Adds a different positive constant to each maximum value in
     $\mathbf{f_{pot}}$ to make sure that there is a single largest value in the vector, prevents leakage when using **cut** to find $\delta_{\mathrm{pot}}$
16 $\llbracket \delta_{\mathrm{pot}} \rrbracket \leftarrow \mathbf{cut}(\llbracket \delta \rrbracket, \mathbf{max}(\llbracket \mathbf{f_{pot}} \rrbracket) == \llbracket \mathbf{f_{pot}} \rrbracket)$
17 $\llbracket \delta_{max} \rrbracket \leftarrow \mathbf{Newton\text{–}Rhapson}(\llbracket \delta_{\mathrm{pot}} \rrbracket)$ //Newton–Rhapson algorithm
18 $\llbracket \mathbf{X1} \rrbracket \leftarrow \llbracket \mathbf{X} \rrbracket$
19 $\llbracket \mathbf{diag} \rrbracket \leftarrow 1/(\llbracket \xi \rrbracket + \llbracket \delta_{max} \rrbracket)$
20 $\llbracket \mathbf{diagMat} \rrbracket \leftarrow \llbracket \mathbf{diag} \rrbracket$ // **diagMat** is a square matrix with **diag** at the diagonal
21 $\llbracket \mathbf{XH} \rrbracket \leftarrow \llbracket \mathbf{X1} \rrbracket \llbracket \mathbf{U_F} \rrbracket \llbracket \mathbf{diagMat} \rrbracket \llbracket \mathbf{U_F} \rrbracket^T$
22 $\llbracket \mathbf{diagXHX} \rrbracket = \mathbf{rowSums}(\llbracket \mathbf{XH} \rrbracket * \llbracket \mathbf{X1} \rrbracket)$
23 $\llbracket \mathbf{XHX}[:, 0] \rrbracket \leftarrow \llbracket \mathbf{diagXHX}[0] \rrbracket$
24 $\llbracket \mathbf{XHX}[:, 3] \rrbracket \leftarrow \llbracket \mathbf{diagXHX}[1 :] \rrbracket$
25 $\llbracket \mathbf{XHX}[:, 1] \rrbracket \leftarrow \mathbf{rowSums}(\llbracket \mathbf{XH} \rrbracket)$
26 $\llbracket \mathbf{XHX}[:, 2] \rrbracket \leftarrow \llbracket \mathbf{XHX}[:, 1] \rrbracket$
27 $\llbracket \mathbf{det} \rrbracket \leftarrow \llbracket \mathbf{XHX}[:, 0] \rrbracket * \llbracket \mathbf{XHX}[:, 3] \rrbracket - \llbracket \mathbf{XHX}[:, 1] \rrbracket * \llbracket \mathbf{XHX}[:, 2] \rrbracket$
28 $\llbracket \mathbf{iXHX}[:, 0] \rrbracket \leftarrow \llbracket \mathbf{XHX}[:, 3] \rrbracket/\llbracket \mathbf{det} \rrbracket$
29 $\llbracket \mathbf{iXHX}[:, 3] \rrbracket \leftarrow \llbracket \mathbf{XHX}[:, 0] \rrbracket/\llbracket \mathbf{det} \rrbracket$
30 $\llbracket \mathbf{iXHX}[:, 1] \rrbracket \leftarrow -\llbracket \mathbf{XHX}[:, 1] \rrbracket/\llbracket \mathbf{det} \rrbracket$
31 $\llbracket \mathbf{iXHX}[:, 2] \rrbracket \leftarrow \llbracket \mathbf{iXHX}[:, 1] \rrbracket$
32 $\llbracket \mathbf{yMat} \rrbracket \leftarrow \llbracket \mathbf{y} \rrbracket$ // **yMat** is an $m \times n$ matrix, each row is equal to $\mathbf{y}$
33 $\llbracket \mathbf{XHy} \rrbracket \leftarrow \mathbf{rowSums}(\llbracket \mathbf{XH} \rrbracket * \llbracket \mathbf{yMat} \rrbracket)$
34 $\llbracket \beta \rrbracket \leftarrow \llbracket \mathbf{iXHX}[:, 2] \rrbracket * \llbracket \mathbf{XHy}[0] \rrbracket + \llbracket \mathbf{iXHX}[:, 3] \rrbracket * \llbracket \mathbf{XHy}[1 :] \rrbracket$
35 $\llbracket \mathbf{yMat2} \rrbracket \leftarrow \llbracket \mathbf{y} \rrbracket$ // **yMat2** is an $n \times n$ matrix, each row is equal to $\mathbf{y}$
36 $\llbracket \mathbf{yU_F} \rrbracket \leftarrow \mathbf{rowSums}(\llbracket \mathbf{yMat2} \rrbracket * \llbracket \mathbf{U_F} \rrbracket)$
37 $\llbracket yHy \rrbracket \leftarrow \mathbf{sum}(\llbracket \mathbf{yU_F} \rrbracket * \llbracket \mathbf{diag} \rrbracket * \llbracket \mathbf{yU_F} \rrbracket)$
38 $\llbracket \mathbf{R1} \rrbracket \leftarrow (\llbracket \mathbf{iXHX}[:, 0] \rrbracket * \llbracket \mathbf{XHy}[0] \rrbracket + \llbracket \mathbf{iXHX}[:, 2] \rrbracket * \llbracket \mathbf{XHy}[1 :] \rrbracket) * \llbracket \mathbf{XHy}[0] \rrbracket$
39 $\llbracket \mathbf{R2} \rrbracket \leftarrow \llbracket \beta \rrbracket * \llbracket \mathbf{XHy}[1 :] \rrbracket$
40 return $\llbracket \mathbf{t} \rrbracket \leftarrow \llbracket \beta \rrbracket/\mathbf{sqrt}(\llbracket \mathbf{iXHX}[:, 3] \rrbracket * (\llbracket yHy \rrbracket - (\llbracket \mathbf{R1} \rrbracket + \llbracket \mathbf{R2} \rrbracket)))$
```

In order to estimate the linear coefficient $\llbracket \beta_k \rrbracket$ of SNP k for every $k = 1, \dots, m$, we need to compute $(\llbracket \mathbf{X_k} \rrbracket \llbracket \mathbf{H} \rrbracket \llbracket \mathbf{X_k} \rrbracket^T)^{-1} \llbracket \mathbf{X_k} \rrbracket \llbracket \mathbf{H} \rrbracket \llbracket \mathbf{y} \rrbracket$ for each k, where

$$[\![\mathbf{H}]\!] = [\![\mathbf{U_F}]\!]\,\mathrm{diag}(([\![\xi]\!] + [\![\delta_{max}]\!])^{-1})[\![\mathbf{U_F}]\!]^T$$

is an $n \times n$ matrix and

$$[\![\mathbf{X_k}]\!] = \begin{pmatrix} 1 & \dots & 1 \\ [\![X_{1k}]\!] & \dots & [\![X_{nk}]\!] \end{pmatrix}.$$

Instead of computing each of these values one at a time, we define the following $(m+1) \times n$ matrix

$$[\![\mathbf{X1}]\!] = \begin{pmatrix} 1 & \dots & 1 \\ [\![X_{11}]\!] & \dots & [\![X_{n1}]\!] \\ \vdots & \ddots & \vdots \\ [\![X_{1m}]\!] & \dots & [\![X_{nm}]\!] \end{pmatrix},$$

and we compute

$$[\![\mathbf{H}]\!] = [\![\mathbf{U_F}]\!]\,\mathrm{diag}(([\![\xi]\!] + [\![\delta_{max}]\!])^{-1})[\![\mathbf{U_F}]\!]^T$$

and

$$[\![\mathbf{XH}]\!] = [\![\mathbf{X1}]\!][\![\mathbf{H}]\!] \ .$$

Note that the first and $(i+1)$th rows of matrix $[\![\mathbf{XH}]\!]$ are equal to the rows of the $2 \times n$ matrix $[\![\mathbf{X_k}]\!][\![\mathbf{H}]\!]$. We continue by computing the entries of the $m \times 4$ matrix $[\![\mathbf{XHX}]\!]$. This matrix contains the entries of each matrix $[\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{X_k}]\!]^T$, more precisely

$$[\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{X_k}]\!]^T = \begin{pmatrix} [\![\mathbf{XHX}[k,0]]\!] & [\![\mathbf{XHX}[k,1]]\!] \\ [\![\mathbf{XHX}[k,2]]\!] & [\![\mathbf{XHX}[k,3]]\!] \end{pmatrix}.$$

As finding the inverse of a 2×2 matrix is easy, we can now create an $m \times 4$ matrix $[\![\mathbf{iXHX}]\!]$, which contains the entries of $([\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{X_k}]\!]^T)^{-1}$ in the kth row. Let $[\![\mathbf{yMat}]\!]$ be an $(m+1) \times n$ matrix for which each row is equal to vector $[\![\mathbf{y}]\!]$. Next, we find the vector $[\![\mathbf{XHy}]\!]$, where the first and $(i+1)$th entries are equal to the entries of the vector $[\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{y}]\!]^T$. The estimate of $[\![\beta_k]\!]$ can now be computed as

$$[\![\beta_k]\!] = [\![\mathbf{iXHX}[k,2]]\!] * [\![\mathbf{XHy}[0]]\!] + [\![\mathbf{iXHX}[k,3]]\!] * [\![\mathbf{XHy}[k+1]]\!] \ .$$

We compute all these estimates in parallel. To find the vector of t-statistics, $[\![\mathbf{t}]\!] = (\frac{[\![\beta_1]\!]}{[\![s_1]\!]}, \dots, \frac{[\![\beta_m]\!]}{[\![s_m]\!]})$, we compute the value $[\![yHy]\!] = [\![\mathbf{y}]\!][\![\mathbf{H}]\!][\![\mathbf{y}]\!]^T$ and the values

$$([\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{y}]\!])^T([\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{X_k}]\!]^T)^{-1}[\![\mathbf{X_k}]\!][\![\mathbf{H}]\!][\![\mathbf{y}]\!] \ ,$$

and use them to find $[\![s_k]\!]$ for all $k = 1, \dots, m$.

3.3. Genomic Control

Algorithm 4 describes the privacy-preserving version of the genomic control algorithm [25]. As input it receives the genotype matrix $[\![\mathbf{X}]\!]$ and the phenotype vector $[\![\mathbf{y}]\!]$. For this privacy-preserving algorithm, we need to manipulate the shape of the data to achieve better performance. As comparison is rather expensive and we would have to perform a significant number of comparisons to count the number of different combinations $\{AA, AB, BB\}$, we will convert the data into the following format: the columns of the matrix $[\![\mathbf{X}]\!]$ indicate individuals and for each SNP we have three rows (for AA, AB and BB), where the value is 1 for the corresponding observation and the values for the other two rows are 0. For the missing SNP values, all three rows are marked as 0.

We would like to compute the entries in Table 2 for each SNP in parallel as parallel operations are optimal in the privacy-preserving setting. For this, we will find the sum of the columns corresponding to the case and control group separately. This gives us six vectors $[\![\mathbf{r}_i]\!]$, $[\![\mathbf{s}_i]\!]$, $i \in \{0,1,2\}$. From these we can compute

$$\begin{aligned}
[\![\mathbf{R}]\!] &= [\![\mathbf{r}_0]\!] + [\![\mathbf{r}_1]\!] + [\![\mathbf{r}_2]\!] \ , \\
[\![\mathbf{S}]\!] &= [\![\mathbf{s}_0]\!] + [\![\mathbf{s}_1]\!] + [\![\mathbf{s}_2]\!] \ , \\
[\![\mathbf{N}]\!] &= [\![\mathbf{R}]\!] + [\![\mathbf{S}]\!] \ , \\
[\![\mathbf{n}_1]\!] &= [\![\mathbf{r}_1]\!] + [\![\mathbf{s}_1]\!] \ , \\
[\![\mathbf{n}_2]\!] &= [\![\mathbf{r}_2]\!] + [\![\mathbf{s}_2]\!] \ .
\end{aligned}$$

With these vectors we can now compute the Cochran–Armitage trend statistics for all SNPs in parallel. We then find the estimate of the variance inflation factor and divide the trend statistics with them, giving us the necessary χ^2 statistics for each SNP. As previously, no values are declassified during the execution of the algorithm.

Algorithm 4: Privacy-preserving genomic control

```
Input: Genotype matrix $[\![\mathbf{X}]\!]$ (private), phenotype vector $[\![\mathbf{y}]\!]$ (private)
Output: Vector of statistics for the $\chi^2$-test stat
$n, m \leftarrow$ shape($[\![\mathbf{X}]\!]$)
$nSNP \leftarrow n/3$
$[\![\mathbf{r}]\!] \leftarrow 0$
$[\![\mathbf{s}]\!] \leftarrow 0$
for $i = 0, \ldots, m-1$ do
    $[\![\mathbf{r}]\!] = [\![\mathbf{r}]\!] + [\![\mathbf{y}[i]]\!] * [\![\mathbf{X}[:, i]]\!]$
    $[\![\mathbf{s}]\!] = [\![\mathbf{s}]\!] + (1 - [\![\mathbf{y}[i]]\!]) * [\![\mathbf{X}[:, i]]\!]$
end
$[\![\mathbf{rMat}]\!] \leftarrow$ reshape($[\![\mathbf{r}]\!], nSNP, 3$)
$[\![\mathbf{sMat}]\!] \leftarrow$ reshape($[\![\mathbf{s}]\!], nSNP, 3$)
$[\![\mathbf{r}_0]\!], [\![\mathbf{r}_1]\!], [\![\mathbf{r}_2]\!] \leftarrow [\![\mathbf{rMat}]\!]$ // $\mathbf{r}_i$ is the $i$th column of rMat
$[\![\mathbf{s}_0]\!], [\![\mathbf{s}_1]\!], [\![\mathbf{s}_2]\!] \leftarrow [\![\mathbf{sMat}]\!]$ // $\mathbf{s}_i$ is the $i$th column of sMat
$[\![\mathbf{R}]\!] \leftarrow [\![\mathbf{r}_0]\!] + [\![\mathbf{r}_1]\!] + [\![\mathbf{r}_2]\!]$
$[\![\mathbf{S}]\!] \leftarrow [\![\mathbf{s}_0]\!] + [\![\mathbf{s}_1]\!] + [\![\mathbf{s}_2]\!]$
$[\![\mathbf{n}_1]\!] \leftarrow [\![\mathbf{r}_1]\!] + [\![\mathbf{s}_1]\!]$
$[\![\mathbf{n}_2]\!] \leftarrow [\![\mathbf{r}_2]\!] + [\![\mathbf{s}_2]\!]$
$[\![\mathbf{N}]\!] \leftarrow [\![\mathbf{R}]\!] + [\![\mathbf{S}]\!]$
$[\![\mathbf{num}]\!] \leftarrow [\![\mathbf{N}]\!] * ([\![\mathbf{N}]\!] * ([\![\mathbf{r}_1]\!] + 2[\![\mathbf{r}_2]\!]) - [\![\mathbf{R}]\!] * ([\![\mathbf{n}_1]\!] + 2[\![\mathbf{n}_2]\!]))^2$
$[\![\mathbf{den}]\!] \leftarrow [\![\mathbf{R}]\!] * ([\![\mathbf{N}]\!] - [\![\mathbf{R}]\!]) * ([\![\mathbf{N}]\!] * ([\![\mathbf{n}_1]\!] + 4[\![\mathbf{n}_2]\!]) - ([\![\mathbf{n}_1]\!] + 2[\![\mathbf{n}_2]\!])^2)$
$[\![\mathbf{m}]\!] \leftarrow [\![\mathbf{den}]\!] != 0$
$[\![\mathbf{den}]\!] \leftarrow$ choose($[\![\mathbf{m}]\!], [\![\mathbf{den}]\!], \mathbf{1}$) // If denominator is equal to 0 replace it with 1
$[\![\mathbf{ca}]\!] \leftarrow [\![\mathbf{num}]\!] / [\![\mathbf{den}]\!]$
$[\![\mathbf{ca\_sorted}]\!] \leftarrow$ quicksort($[\![\mathbf{ca}]\!]$)
$[\![\lambda]\!] \leftarrow$ max($[\![\mathbf{ca\_sorted}[nSNP/2]]\!]/0.455, 1$)
return stat $\leftarrow [\![\mathbf{ca}]\!] / [\![\lambda]\!]$
```

3.4. Performance Measurements

All of the algorithms running on SHAREMIND MPC are implemented in the SecreC programming language [34,35]. The trusted execution environment versions are implemented in C/C++ and require the SHAREMIND HI runtime. The source code for each is available for download as additional material. The data that we used to run and measure our implementation were taken from the NCBI Gene Expression Omnibus (GEO) (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE44974, accessed on 19 August 2021). It is available for download. However, the contents of the dataset do not affect the performance results as the running times of the privacy-preserving algorithms are data-independent by design; otherwise this would leak information about the dataset.

As discussed in Section 2.3, secure multi-party computation introduces a level of complexity to already complex algorithms. In addition to designing privacy-preserving versions of frequently used GWAS algorithms, we also show that they are implementable and run on a volume of data that would be used for realistic genome-wide association studies. We implemented our algorithms using the privacy-preserving computation platform SHAREMIND. For MPC, we used the three-party protocol suite in the passive adversary model [36]. For HI, we used the Intel Software Guard Extensions (SGX).

First we present the performance results of the MPC solution, then the TEE solution, and finally we compare the two approaches. The performance of the MPC algorithms was benchmarked using three servers with Intel Xeon E5-2640 processors, 128 GB of memory and dedicated 10 Gb/s connections. In the benchmark tables, we show running times for the different subtasks of the algorithms, thus giving a better overview of where optimisation can be attempted.

Table 3 presents benchmark results for our privacy-preserving EIGENSTRAT algorithm. Table preparation includes the time needed for reading the table and normalising it. Stratification control includes the time needed for finding the kinship matrix, subtracting the genotype and phenotype principal components. We looked at the running times for 1500, 2000, 5000 and 20,000 SNPs for 217 donors, and then also increased the number of donors to look at 2000 SNPs for 434 and 868 donors. The size of the case and control groups does not influence running times.

Table 3. Benchmark results for privacy-preserving EIGENSTRAT.

Subtask	**1500 SNPs 217 Donors**	**2000 SNPs 217 Donors**	**5000 SNPs 217 Donors**	**20,000 SNPs 217 Donors**	**2000 SNPs 434 Donors**	**2000 SNPs 868 Donors**
Table preparation	67.4 s	85.4 s	218.4 s	865.1 s	161.6 s	347.0 s
GS-PCA	182.8 s	187.9 s	183.3 s	704.8 s	685.8 s	2606.0 s
Stratification control	1365.9 s	1849.5 s	4520.2 s	18,684.3 s	6214.4 s	22,112.0 s
Test statistics	136.2 s	174.5 s	436.6 s	1681.7 s	344.2 s	675.4 s
Total	1752.4 s (29.2 min)	2297.2 s (38.3 min)	5358.4 s (89.3 min)	21,413.2 s (5.95 h)	7406.0 s (2.06 h)	25,740.4 s (7.15 h)

For our privacy-preserving EIGENSTRAT, the computation of the kinship matrix proved to be the biggest computational overhead, scaling linearly with respect to the number of SNPs and quadratically with respect to the size of the sample in our data table. The GS-PCA algorithm, used for finding the top eigenvectors of the kinship matrix, also scales quadratically with respect to the size of the sample. This is because the kinship matrix is an $n \times n$ matrix, where n is the number of people in our sample. As such, the running time of the GS-PCA algorithm does not depend on the number of SNPs. Other computational steps scale linearly with respect to the dimensions of the input matrix.

Table 4 presents benchmark results for our privacy-preserving FastPCA randomised algorithm. Table preparation includes the time needed for reading the table and normalising it. Stratification control includes the time needed for subtracting the genotype and phenotype principal components. Similarly to EIGENSTRAT, we looked at the running times for 1500, 2000, 5000, and 20,000 SNPs for 217 donors, and then also increased the number of donors to look at 2000 SNPs for 434 and 868 donors. The size of the case and control groups does not influence running times.

The main benefit of FastPCA is that it does not require computing the kinship matrix, and, therefore, it scales linearly with respect to both the number of SNPs and the sample size. In addition, eigendecomposition is only applied to a small $(k + p) \times (k + p)$ matrix, which offers clear speed-ups with respect to the GS-PCA approach.

Table 4. Benchmark results for privacy-preserving FastPCA.

Subtask	1500 SNPs 217 Donors	2000 SNPs 217 Donors	5000 SNPs 217 Donors	20,000 SNPs 217 Donors	2000 SNPs 434 Donors	2000 SNPs 868 Donors
Table preparation	75.7 s	102.1 s	247.9 s	1000.8 s	198.1 s	393.9 s
PCA using random matrix theory	2482.0 s	3290.9 s	7920.7 s	31,355.6 s	6194.8 s	11,900.2 s
Stratification control	216.3 s	293.7 s	707.3 s	2800.8 s	564.9 s	1139.3 s
Test statistics	159.1 s	204.9 s	505.6 s	1998.5 s	407.9 s	815.6 s
Total	2933.0 s (48.9 min)	3891.5 s (64.9 min)	9381.5 s (2.61 h)	37,155.7 s (10.32 h)	7365.7 s (2.04 h)	14,249.0 s (3.96 h)

Table 5 presents benchmark results for our privacy-preserving EMMAX algorithm. Table preparation includes the time needed for reading the table and normalising it. The maximum likelihood times include everything in Algorithm 3 after the call to GS-PCA and before computing the test statistics. Most of the time in this group is spent on computing $[\![\mathbf{XHX}]\!]$. As the privacy-preserving EMMAX algorithm is significantly slower than the privacy-preserving PCA algorithm, we looked at the running times for 1000, 5000 and 20,000 SNPs for 217 donors and then also looked at 1000 SNPs for 100 and 434 donors. As can be seen from Table 5, the running time for 1000 SNPs and 434 donors is already more than 36 h, and we did not test the algorithm any further. The size of the case and control groups does not influence running times.

Table 5. Benchmark results for privacy-preserving EMMAX.

Subtask	1000 SNPs 217 Donors	5000 SNPs 217 Donors	20,000 SNPs 217 Donors	1000 SNPs 100 Donors	1000 SNPs 434 Donors
Table preparation	44.7 s	239.3 s	891.0 s	24.7 s	89.8 s
Kinship matrix	700.0 s	3420.2 s	13,988.5 s	173.3 s	2844.0 s
GS-PCA	14,241.6 s	14,209.2 s	14,239.3 s	1912.8 s	104,067.3 s
Maximum likelihood	4934.3 s	21,381.8 s	81,961.9 s	1147.5 s	23,185.7 s
Test statistics	0.4 s	1.7 s	7.6 s	0.4 s	0.4 s
Total	19,920.9 s (5.53 h)	39,252.5 s (10.90 h)	111,088.3 s (30.86 h)	3258.7 s (54.3 min)	130,187.1 s (36.2 h)

Similarly to EIGENSTRAT, our EMMAX implementation computes the kinship matrix, which scales linearly with respect to the number of SNPs and quadratically with respect to the size of the sample. We then use the GS-PCA algorithm to compute the eigenvectors and eigenvalues of the kinship matrix. However, for EMMAX, we need the entire eigendecompostion instead of a few eigenvectors. This means that the process takes significantly longer compared to the privacy-preserving PCA. It is clear that as the size of the genotype matrix grows, the computation of the $[\![\mathbf{XHX}]\!]$ matrix quickly overshadows the computational effort of the rest of the algorithm.

Table 6 presents benchmark results for our privacy-preserving genomic control algorithm. We performed these computations for the usual 5000 SNPs and 217 donors but then went on to test with far larger datasets than the PCA experiments. Here, most of the running time is spent on table preparation, which includes table reading and computing the genotype distribution table for the Cochran–Armitage tests. Note that due to the way data are managed in this algorithm, 300,000 SNPs will take up 900,000 rows in the dataset.

Benchmark details for the HI versions of PCA and EMMAX are given in Table 7. Unlike the MPC benchmarks, these results are given in milliseconds. The SHAREMIND HI tests were run on a PC with i7-7700 CPU and 16 GB RAM.

Table 6. Benchmark results for privacy-preserving genomic control.

Subtask	5000 SNPs 217 Donors	300,000 SNPs 217 Donors	300,000 SNPs 434 Donors	300,000 SNPs 661 Donors
Table preparation	33.8 s	1912.3 s	3619.5 s	5451.1 s
Cochran–Armitage test	0.3 s	24.1 s	23.5 s	23.8 s
Stratification control	0.9 s	83.3 s	83.4 s	83.6 s
Total	34.9 s (0.6 min)	2019.7 s (33.7 min)	3726.4 s (62.1 min)	5558.5 s (92.6 min)

Table 7. Benchmark results for the HI implementations of EIGENSTRAT and EMMAX.

Subtask	EIGENSTRAT				EMMAX	
	2000 SNPs 217 Donors	5000 SNPs 217 Donors	20,000 SNPs 217 Donors	2000 SNPs 868 Donors	1000 SNPs 217 Donors	5000 SNPs 217 Donors
Table preparation	101 ms	253 ms	1600 ms	272 ms	28 ms	215 ms
Kinship matrix	219 ms	606 ms	2642 ms	3552 ms	111 ms	609 ms
GS-PCA	46 ms	53 ms	69 ms	735 ms	4202 ms	4220 ms
Stratification control/	443 ms	1337 ms	15675 ms	2782 ms		
Maximum likelihood					26 ms	25 ms
Test statistics	74 ms	175 ms	2489 ms	442 ms	409 ms	2076 ms
Total	883 ms	2424 ms	22475 ms	7783 ms	4776 ms	7145 ms

With the genomic control algorithm for SHAREMIND HI, the running time of 15,112 milliseconds for 300,000 SNPs and 200 donors was mostly spent on reading input data (1 ms of the total running time was spent on other computations). For comparison, we performed the experiments on the same data structures as for MPC. As discussed, this involves having three rows for each SNP. As reading data is an expensive operation in HI but comparing strings is not, the HI version can be optimised for real-world data encodings. After the data was read, the computations themselves took altogether one millisecond. Therefore, we did not run these tests for fewer SNPs.

Table 8 presents the comparison of benchmark results for the SHAREMIND MPC and HI implementations of privacy-preserving GWAS algorithms with stratification control. As expected, the algorithms ran nearly instantaneously on SHAREMIND HI. The table also gives the running times if HI was 100 times slower, an artificial estimate for if we were to take more special measures to avoid side-channel attacks. For genomic control, this would still be around 20 s for reading input data, because the computation times themselves would still not exceed seconds.

Table 8. Comparison of the SHAREMIND MPC and HI implementations of privacy-preserving GWAS algorithms with stratification control.

Experiment	Data Size	MPC	HI	If HI Was 100 Times Slower
EIGENSTRAT	5000 SNPs 200 donors	5358.4 s (89.3 min)	2.42 s	242 s
EMMAX	5000 SNPs 200 donors	87,400 s (24.3 h)	7.14 s	714 s
Genomic control	300,000 SNPs 200 donors	2096.5 s (35 min)	15.11 s (reading input data)	$\sim$20 s (reading input data)

4. Discussion

Feasibility. We showed how to adapt complex stratification control algorithms for genome-wide association studies to secure computing. We especially demonstrate the most complex PCA algorithms implemented in the literature on large datasets. In addition, we adapted the algorithms to trusted execution environments, showcasing how the side-channel security of the proposed algorithms can benefit both MPC and TEEs.

Security. Both secure multi-party computation and trusted execution environments can offer cryptographic security; however, both solutions have their advantages and drawbacks. For MPC, it requires the service providers to set up at least two servers. More complex algorithms might not be implementable or might not finish execution in this environment. The computations that do finish introduce a major computation and communication overhead. However, when the input party trusts their data to the computing parties, they do not need to trust one single party but can trust that the parties do not collaborate. Moreover, to be certain, an input party can become a computing party and thus gain more confidence.

The trusted execution environment does not require such a complicated setup and collaboration from different parties. There is only one server and everyone can import their data in encrypted format. In addition, the environment does not introduce a significant computational or communication overhead. However, the input parties must trust the trusted execution environment provider (Intel). If something in the enclave fails, if a critical vulnerability is discovered, all data could be compromised. Designers of TEE-based data analysis must create a data lifecycle that reduces these risks.

Performance. It is clear that in the MPC setting, the running times of EMMAX are not feasible for real-world use for larger data volumes. It is a complex iterative algorithm that includes a lot of floating-point matrix multiplications. We fear that further optimisations would not yield a significant speed-up. However, genomic control shows that it is feasible to also carry out privacy-preserving GWAS with stratification control using MPC. The privacy-preserving EIGENSTRAT remains on the border of feasibility. However, for all four algorithms, we show that the computations can be run and be completed.

For the solution based on the Intel Software Guard Extensions (SGX), it is clear that the computations are efficient and run in similar time to the algorithms that do not use secure computing. However, even if side channel attack mitigations make the computation 100 times slower, it will outperform MPC, according to our experiments. Thus, further comparison of the security properties of real-world deployments of these technologies will be needed.

Author Contributions: Conceptualisation, methodology: A.O. and S.L.; MPC algorithms, MPC implementation, MPC benchmarking, writing—original draft: A.O.; HI algorithms, HI implementation: V.S.; HI implementation, HI benchmarking: J.R.; validation, writing—original draft, writing—review and editing, project administration, supervision: L.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (http://cybersec4europe.eu, accessed on 19 August 2021). This research has also been supported by the European Regional Development Fund through the Estonian Centre of Excellence in ICT Research (EXCITE), and by the Estonian Research Council through grants PRG49 and PRG920.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that we used to run and measure our implementation were taken from the NCBI Gene Expression Omnibus (GEO) (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE44974, accessed on 19 August 2021). It is available for download. However, the contents of the dataset do not affect the performance results as the running times of the privacy-preserving algorithms are data-independent by design. Therefore, any other dataset (real or synthetic) can be used to reproduce the benchmark results in this paper.

Acknowledgments: The authors wish to thank Dan Bogdanov for their support and suggestions for this paper, and Armin Daniel Kisand for their insights on trusted execution environments and Intel SGX.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Hartl, D.L.; Clark, A.G. *Principles of Population Genetics*, 4th ed.; Sinauer Associates: Sunderland, MA, USA, 2006.
2. Hellwege, J.N.; Keaton, J.M.; Giri, A.; Gao, X.; Velez Edwards, D.R.; Edwards, T.L. Population Stratification in Genetic Association Studies. *Curr. Protoc. Hum. Genet.* **2017**, *95*, 1.22.1–1.22.23. [CrossRef]
3. Campbell, C.D.; Ogburn, E.L.; Lunetta, K.L.; Lyon, H.N.; Freedman, M.L.; Groop, L.C.; Altshuler, D.; Ardlie, K.G.; Hirschhorn, J.N. Demonstrating stratification in a European American population. *Nat. Genet.* **2005**, *37*, 868. [CrossRef]
4. European Data Protection Board. Recommendations 01/2020 on Measures that Supplement Transfer Tools to Ensure Compliance with the EU Level of Protection of Personal Data. 2020. Available online: https://edpb.europa.eu/our-work-tools/public-consultations-art-704/2020/recommendations-012020-measures-supplement-transfer_en (accessed on 19 August 2021).
5. European Data Protection Supervisor. Preliminary Opinion 8/2020 on the European Health Data Space. 2020. Available online: https://edps.europa.eu/data-protection/our-work/publications/opinions/preliminary-opinion-82020-european-health-data-space_en (accessed on 19 August 2021).
6. Kamm, L.; Bogdanov, D.; Laur, S.; Vilo, J. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics* **2013**, *29*, 886–893. [CrossRef] [PubMed]
7. Constable, S.D.; Tang, Y.; Wang, S.; Jiang, X.; Chapin, S. Privacy-preserving GWAS analysis on federated genomic datasets. *BMC Med. Inform. Decis. Mak.* **2015**, *15*, S2. [CrossRef] [PubMed]
8. 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. Bioinform.* **2018**, *15*, 1427–1432. [CrossRef] [PubMed]
9. Bonte, C.; Makri, E.; Ardeshirdavani, A.; Simm, J.; Moreau, Y.; Vercauteren, F. Towards practical privacy-preserving genome-wide association study. *BMC Bioinform.* **2018**, *19*, 537. [CrossRef]
10. Cho, H.; Wu, D.J.; Berger, B. Secure genome-wide association analysis using multiparty computation. *Nat. Biotechnol.* **2018**, *36*, 547–551. [CrossRef]
11. Tkachenko, O.; Weinert, C.; Schneider, T.; Hamacher, K. Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS'18), Incheon, Korea, 4–8 June 2018; ACM: New York, NY, USA, 2018; pp. 221–235.
12. Bellafqira, R.; Ludwig, T.E.; Niyitegeka, D.; Génin, E.; Coatrieux, G. Privacy-Preserving Genome-Wide Association Study for Rare Mutations—A Secure FrameWork for Externalized Statistical Analysis. *IEEE Access* **2020**, *8*, 112515–112529. [CrossRef]
13. Poddar, R.; Kalra, S.; Yanai, A.; Deng, R.; Popa, R.A.; Hellerstein, J.M. Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Online, 11–13 August 2021; USENIX Association: Vancouver, BC, Canada, 2021.
14. Zhang, Y.; Dai, W.; Jiang, X.; Xiong, H.; Wang, S. FORESEE: Fully Outsourced secuRe gEnome Study basEd on homomorphic Encryption. *BMC Med. Inform. Decis. Mak.* **2015**, *15*, S5. [CrossRef]
15. Wang, S.; Zhang, Y.; Dai, W.; Lauter, K.E.; Kim, M.; Tang, Y.; Xiong, H.; Jiang, X. HEALER: homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS. *Bioinformatics* **2016**, *32*, 211–218. [CrossRef]
16. Chen, F.; Yang, H.; Kim, J.; Ohno-Machado, L.; Ding, S.; Jiang, X.; Wang, S.; Fox, D.; Lauter, K.; Lu, Y.; et al. PRINCESS: Privacy-protecting Rare disease International Network Collaboration via Encryption through Software guard extensionS. *Bioinformatics* **2016**, *33*, 871–878. [CrossRef]
17. Asvadishirehjini, A.; Kantarcioglu, M.; Malin, B. A Framework for Privacy-Preserving Genomic Data Analysis Using Trusted Execution Environments. In Proceedings of the 2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), Atlanta, GA, USA, 28–31 October 2020; pp. 138–147.
18. Kockan, C.; Zhu, K.; Dokmai, N.; Karpov, N.; Kulekci, M.O.; Woodruff, D.P.; Sahinalp, S.C. Sketching algorithms for genomic data analysis and querying in a secure enclave. *Nat. Methods* **2020**, *17*, 295–301. [CrossRef] [PubMed]
19. Pascoal, T.; Decouchant, J.; Boutet, A.; Veríssimo, P. DyPS: Dynamic, Private and Secure GWAS. In Proceedings of the Privacy Enhancing Technologies (PoPETS), Online, 12–16 July 2021; Volume 2, pp. 214–234.
20. Price, A.L.; Patterson, N.J.; Plenge, R.M.; Weinblatt, M.E.; Shadick, N.A.; Reich, D. Principal Components Analysis Corrects for Stratification in Genome-Wide Association Studies. *Nat. Genet.* **2006**, *38*, 904–909. [CrossRef]
21. Simmons, S.; Sahinalp, C.; Berger, B. Enabling Privacy-Preserving GWASs in Heterogeneous Human Populations. *Cell Syst.* **2016**, *3*, 54–61. [CrossRef]
22. Mittos, A.; Malin, B.; Cristofaro, E.D. Systematizing Genomic Privacy Research—A Critical Analysis. *arXiv* **2017**, arXiv:abs/1712.02193.

23. Galinsky, K.J.; Bhatia, G.; Loh, P.R.; Georgiev, S.; Mukherjee, S.; Patterson, N.J.; Price, A.L. Fast Principal-Component Analysis Reveals Convergent Evolution of ADH1B in Europe and East Asia. *Am. J. Hum. Genet.* **2016**, *98*, 456–472. [CrossRef] [PubMed]
24. Kang, H.M.; Sul, J.H.; Service, S.K.; Zaitlen, N.A.; Kong, S.Y.; Freimer, N.B.; Sabatti, C.; Eskin, E. Variance component model to account for sample structure in genome-wide association studies. *Nat. Genet.* **2010**, *42*, 348–354. [CrossRef]
25. Devlin, B.; Roeder, K. Genomic Control for Association Studies. *Biometrics* **1999**, *55*, 997–1004. [CrossRef] [PubMed]
26. Bogdanov, D. Sharemind: Programmable Secure Computations with Practical Applications. Ph.D. Thesis, University of Tartu, Tartu, Estonia, 2013.
27. Cramer, R.; Damgård, I.; Nielsen, J. *Secure Multiparty Computation and Secret Sharing;* Cambridge University Press: New York, NY, USA, 2015.
28. Archer, D.W.; Bogdanov, D.; Lindell, Y.; Kamm, L.; Nielsen, K.; Pagter, J.I.; Smart, N.P.; Wright, R.N. From Keys to Databases—Real-World Applications of Secure Multi-Party Computation. *Comput. J.* **2018**, *61*, 1749–1771. [CrossRef]
29. Hastings, M.; Hemenway, B.; Noble, D.; Zdancewic, S. SoK: General Purpose Compilers for Secure Multi-Party Computation. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 1220–1237.
30. Randmets, J. *An Overview of Vulnerabilities and Mitigations of Intel SGX Applications*; Technical Report D-2-116; Cybernetica AS: Tallinn, Estonia, 2021.
31. Bogdanov, D.; Kamm, L.; Laur, S.; Sokk, V. Rmind: A tool for cryptographically secure statistical analysis. *IEEE Trans. Depend. Secur. Comput.* **2016**, *15*, 481–495. [CrossRef]
32. Bogdanov, D.; Laur, S.; Talviste, R. A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In Proceedings of the 19th Nordic Conference on Secure IT Systems (NordSec 2014), Tromsø, Norway, 15–17 October 2014; LNCS; Springer: Cham, Switzerland, 2014; Volume 8788, pp. 59–74.
33. Golub, G.H.; Van Loan, C.F. *Matrix Computations*, 4th ed.; John Hopkins University Press: Baltimore, MD, USA, 2013.
34. Laud, P.; Randmets, J. A Domain-Specific Language for Low-Level Secure Multiparty Computation Protocols. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (ACM 2015), Denver, CO, USA, 12–16 October 2015; pp. 1492–1503.
35. Randmets, J. Programming Languages for Secure Multi-Party Computation Application Development. Ph.D. Thesis, University of Tartu, Tartu, Estonia, 2017.
36. Bogdanov, D.; Niitsoo, M.; Toft, T.; Willemson, J. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Secur.* **2012**, *11*, 403–418. [CrossRef]