# Privacy-Preserving Genome-Wide Association Study is Practical

Charlotte Bonte[2,4], Eleftheria Makri[2,3,4], Amin Ardeshirdavani[1], Jaak Simm[1], Yves Moreau[1,5], Frederik Vercauteren[2,5]

[1] STADIUS KU Leuven
[2] imec-Cosic, Dept. Electrical Engineering, KU Leuven
[3] ABRR Saxion University of Applied Sciences
[4] Joint first authors
[5] Joint last authors

**Abstract.** The deployment of Genome-wide association studies (GWASs) requires genomic information of a large population to produce reliable results. This raises significant privacy concerns, making people hesitate to contribute their genetic information to such studies. We propose two provably secure solutions to address this challenge: (1) a somewhat homomorphic encryption approach, and (2) a secure multiparty computation approach. Unlike previous work, our approach does not rely on adding noise to the input data, nor does it reveal any information about the patients. Our protocols calculate the $\chi^2$ statistic in a privacy-preserving manner, without revealing any information other than the significance of the statistic; hence not even the statistic value itself. We significantly increased the efficiency of our protocols by introducing a new masking technique to perform the secure comparison. Our implementations demonstrated that both approaches are efficient. The secure multiparty computation technique completes its execution in approximately 2 ms for data contributed by one million subjects.

## 1 Introduction

The goal of a genome-wide association study (GWAS) is to identify genetic variants that are associated with traits. Typically, GWASs focus on the associations between single nucleotide polymorphisms (SNPs), and a particular disease. A common approach is to divide the population into a disease, and a healthy group based on whether the individual has the particular disease. Each individual gives a sample DNA from which millions of genetic variants (i.e., SNPs) are identified. If a variant is more frequent in individuals with the disease it will be associated with the specific genetic disorder. The final assessment for each SNP is carried out using statistical methods, such as the $\chi^2$ test.

In recent years the development of cheap next generation sequencing (NGS) has greatly boosted the number of GWASs. Having a large population size is

This work was supported by the European Commission under the ICT programme with contract H2020-ICT-2014-1 644209 HEAT.

crucial for a GWAS because it allows to improve the accuracy of identified associations, especially for *rare* genetic disorders. Another promising direction is the creation of *distributed genomic databases* that enables pooling of data from many hospitals, and research centers, further increasing the population sizes of the studies by 10-50 times [ASD+14], [fGH16]. Several such distributed databases have recently been proposed, including NGS-Logistics [ASD+14], Elixir, and GA4GH Beacon [fGH16].

However, privacy issues with such distributed databases are paramount. The privacy concerns arise even if only aggregated SNP frequencies at the population level (necessary for $\chi^2$ testing) are disclosed. This was first noticed by Homer et al. [HSR+08]. Even access to *aggregate* statistics of SNPs, such as frequency, allows a researcher to identify whether an individual was present in the study. This is a major privacy concern, because the aggregated statistics cover both SNP, and disease data of the individuals. The attack by Homer et al. [HSR+08] can firstly reveal whether the person participated in the GWAS, and secondly whether the person was in the disease or the healthy group.

A classical solution to the privacy problem involves a trusted third party who first collects both the SNP, and the trait data, then carries out the statistical test, and finally either a) only reveals the very few SNPs that have statistically significant association or b) reveals all aggregate data on SNPs but masks them with sufficient noise to guarantee *differential privacy*, for example by using a differentially private $\chi^2$ test, such as the ones proposed by Uhlerop et al. [USF13] or by Simmons and Berger [SB16].

However, setting up such a trusted third party has significant legal, and technical difficulties given the sensitive nature of the underlying data. Additionally, the trusted third party approach does not provide defense against malicious agents or operating system bugs. This can lead to the disclosure of the aggregated SNP data or trait data, which as mentioned before, can be used to reveal personal details of the participants of the study.

To solve this issue we propose a cryptographic approach, where the trusted third party is replaced by a *privacy-preserving* system, which receives the input in encrypted (protected) form from a set of distributed parties (e.g., hospitals), performs the $\chi^2$ test, and only discloses whether the current test is significant or not. Since nothing except the final answer is revealed during the execution of our protocols, the proposed system enjoys various security guarantees, even against malicious agents who gain access to the servers executing the system. Even though a cryptographically secure system for GWAS-like problems has been proposed before by Kamm et al. [KBLV13], its application has been limited to small datasets only, containing up to one thousand patients, reaching only tens of SNP-trait hypotheses per day (based on Table 4 in [KBLV13]). In contrast, our work can scale up to *millions of patients*, and perform *millions or tens of millions of hypothesis tests* per day. This enables the execution of a large-scale distributed GWAS without a trusted central third party.

To allow for a privacy-preserving system addressing our challenges, we propose two secure approaches: one based on homomorphic encryption (HE), and

one based on multiparty computation (MPC). We also compare their security guarantees, and their efficiency in terms of execution time of practical implementations. Homomorphic encryption refers to a set of cryptographic tools that allow certain computations to take place in the encrypted domain, while the resulting ciphertext, when decrypted, is the expected (correct) result of operations on the plaintext data. Secure multiparty computation aims at allowing a similar functionality, amongst several mutually distrusted parties, who wish to compute a function without revealing their private inputs. With the latter approach communication between the computing parties is required for the execution of the cryptographic protocols.

An important feature of our work is that it provides provable security guarantees (unlike the differential privacy techniques). In the MPC setting, there are two main security models used, offering passive, or active security, respectively. *Passive security*, also known as security in the semi-honest model, assumes that the protocol participants are honest-but-curious. This means that they are trying to collect as much information as possible from the protocol execution, but they do follow the protocol instructions honestly. *Active security*, also known as malicious security, offers stronger security guarantees, assuming that adversaries or corrupted protocol participants may arbitrarily deviate from the protocol instructions. In both security models we can build protocols assuming an honest majority of the protocol participants, or a dishonest majority. Our solution offers the highest security guarantees being built in the malicious model, with dishonest majority.

Specifically, we make the following contributions:

- We propose the first *somewhat homomorphic encryption* approach to withstand GWAS attacks such as the ones described by Homer et al. [HSR+08]
- We develop a *multiparty computation* solution for GWAS that is efficient for realistic sample sizes
- We propose a new masking technique to allow efficient secure comparisons
- We compare on a real-life application the security, and efficiency of HE and MPC
- We demonstrate the practicality of our solutions, based on their short running times, which are in the range of 1.4-2.2 ms for the MPC approach
- We show that our solution scales logarithmically in the number of subjects contributing their genetic information, allowing us to treat current full-scale GWAS, and being able to scale to larger (future) GWASs for millions of people

In the following section we discuss related work. Next we review the general scenario for which we propose the two aforementioned solutions. In Section 4 the solution based on homomorphic encryption is detailed, together with its technical description, implementation details, and performance analysis. In Section 5 the secure multiparty computation approach is explained in a similar fashion as the homomorphic encryption approach. Section 6 concludes our work, and provides a comparison of the two approaches.

## 2 Related Work

### 2.1 Homomorphic Encryption Approach

There has already been some work on using homomorphic encryption to preserve the privacy of the patients while performing statistics on genome data. Kim et al. [KL15] present the computation of minor allelle frequencies and the $\chi^2$ statistic with the use of the homomorphic BVG and YASHE encryption schemes. They use a specific encoding technique to improve on the work of Lauter et al. [LLAN14]. However, they only compute the allele counts homomorphically, and execute the other operations on the decrypted data.

Another work on GWASs using fully homomorphic encryption was published by Lu et al. [LYS15]. They also start from encrypted genotype/phenotype information that is uploaded to a cloud for each person seperately. Then they perform the minimal operations necessary to provide someone with access to the decryption key with the necessary values to construct the contingency table for the requested case based on the data present on the cloud. Hence, when performing a request, the scientist gets three encrypted values, and based on those he can, after decryption, reconstruct the contingency table, and compute the $\chi^2$ statistic in the clear. These solutions are not resistant to attacks like the one described by Homer et al. [HSR+08]. Our solution improves on these previous works by performing the $\chi^2$ computation in the encrypted domain and revealing only whether or not the $\chi^2$ value is significant for this case, which makes the previously mentioned attacks impossible.
Sadat et al. [SAM+17] propose a hybrid system called SAFETY to compute various statistical values over genomic data. This hybrid system consists of a combination of the partially homomorphic Paillier scheme with the secure hardware component of Intel Software Guard Extensions (Intel SGX) to ensure both high efficiency, and privacy. With this hybrid system they propose a more efficient way to get the total counts of all patients for a specific case. By using the additive property of the homomorphic Paillier scheme, they reduce the computational overhead of decrypting all individual encrypted outputs received from the different servers. Afterwards it uses the Intel SGX component to perform the $\chi^2$ computations. Even though the results of this system scale well for increasing amount of servers that provide data for the computation, the system does not provide the same functionality as our solution. Sadat et al. [SAM+17] mention that the only privacy guarantee for the final computation result against the attack described by [HSR+08] is the assumption that the researcher decrypting the result is semi-honest. This is the main difference with our work: with our solution the reasearcher will only find out whether this case is significant, and not the $\chi^2$ value itself.

### 2.2 Secure Multiparty Computation Approach

Kamm et al. [KBLV13] propose a solution to address the privacy challenges in genome-wide association studies. Their application scenarios, much like ours, fo-

cus on large data collections from several biobanks, and their solutions are based on the same fundamental techniques as ours. However, Kamm et al.'s [KBLV13] setting requires all raw genotype, phenotype, and clinical data to be entered to the secure shared database. To the contrary, our setting assumes that only the aggregate values, necessary to identify the significance of a gene-disease relationship (i.e., the contingency tables recording the counts of genotypes vs. phenotypes), are contributed by each biobank. This is a simpler, and more realistic setting, which not only is likely to be implemented in the near future, but also alleviates the computational cost of the proposed solutions. Unlike the approach of Kamm et al. [KBLV13], and the alternatives that they suggest, our solution achieves active security with dishonest majority (contrary to the semi-honest security suggested). This means that our protocols tolerate dishonest behavior by the majority of the computing parties, while preserving privacy, and still guarantee the correctness of accepted results. Kamm et al.'s protocols assume that the computing parties -the biobanks- cannot be corrupted, which we consider to be a strong assumption.

Constable et al. [CTW+15] present a garbled-circuit based MPC approach to perform GWAS. Their solution can compute in a privacy-preserving manner the minor allele frequency (MAF), and the $\chi^2$ statistic. Similarly to Kamm et al.'s [KBLV13] work, the framework of Constable et al. [CTW+15] requires the raw genotype, and phenotype data, increasing the workload of the proposed privacy-preserving system. In contrast to our solution, which can scale to hundreds of medical centers contributing data to the GWAS, the solution of Constable et al. [CTW+15] only works for two medical centers. Despite the strong security guarantees that our approach offers, which generally presents itself as a tradeoff to efficiency, our proposal is faster than that of Constable et al. [CTW+15]. This is also due to the fact that we have optimized the computations of the $\chi^2$ statistic, in such a way that the expensive computations in the privacy-preserving domain, are avoided to the maximum extent possible.

Zhang et al. [ZBA15] propose a secret-sharing based MPC approach to solve the same GWAS problem as Constable et al. [CTW+15]. Although Zhang et al.'s solution can scale to more than two medical centers contributing data to the GWAS, the approach has the same inherent limitations (e.g., requiring raw genomic data as input) that their application scenario incurs. None of the related works has considered protecting the aggregate statistic result of the private computation, which –as Homer et al. [HSR+08] showed– can be used to breach an individual's privacy. We are the first to additionally protect the aggregate statistic result, while at the same time allowing for a public list to be created, showing which SNPs are significant for a certain disease.

## 3 Distributed GWAS Scenario

In this paper we aim at identifying which mutations are linked to which diseases, without compromising the privacy of the patients. Specifically, there are $K$ centers (hospitals) who each have genotype (SNP), and phenotype (trait) data. For

a single genotype-phenotype pair a center $k$ has a $2 \times 2$ contingency table[6] of the counts of patients for all 4 possible combinations of genotype, and phenotype (see Table 1). The goal is to perform a privacy-preserving computation that adds together all contingency tables from individual centers, then computes the Pearson's $\chi^2$ test statistic [Pea00], and finally reveals a boolean value indicating whether the computed statistic is larger than a predetermined significance threshold $t$. This threshold is chosen based on the $p$-value, and the correction for multiple hypothesis testing. For example, using significance level 0.01 with Bonferroni correction for 10 million tests results in $t = 37.3$, and for 100 million tests $t = 41.8$.

We propose two different methods for carrying out the $\chi^2$ test without disclosing the input, and intermediate values. The first method performs all computations on *homomorphically* encrypted data, while the second applies techniques of *secure multiparty computation* to achieve the same goal. Both methods follow the same general outline, presented below. The first step is to encrypt (or secret share) all the input tables from the centers, and securely compute the aggregate contingency table

$$O_{ij} = \sum_{k=1}^{K} O_{ij}^{(k)}, \tag{1}$$

where $O_{ij}^{(k)}$ is the data from $k$-th center. This step is straightforward in both methods.

Next to determine the significance of the relation between a mutation, and a disease, we calculate the Pearson's $\chi^2$ test statistic [Pea00] on the aggregated contingency table $O$, and check whether this value is above the threshold $t$. The Pearson's $\chi^2$ statistic is given by the following formula:

$$\chi^2 = \sum_{i,j \in \{1,2\}} \frac{(O_{ij} - model_{ij})^2}{model_{ij}}, \tag{2}$$

where $model_{ij} = (RT_i \cdot CT_j)/N$ with $RT_i = O_{i,1} + O_{i,2}$ being the row total, $CT_j = O_{1,j} + O_{2,j}$ being the column total, and $N$ the total number of patients.

| | $phenotype$ | $\neg phenotype$ | |
|---|---|---|---|
| $genotype$ | $O_{1,1}$ | $O_{1,2}$ | $RT_1$ |
| $\neg genotype$ | $O_{2,1}$ | $O_{2,2}$ | $RT_2$ |
| | $CT_1$ | $CT_2$ | $N$ |

Table 1: Representation of a contingency table

[6] Our method can be also extended to contingency tables of larger size.

Since division is a costly operation in both the homomorphic domain, and secret shared domain, we will rewrite the formula of the $\chi^2$ statistic as follows:

$$\chi^2 = \frac{RT_1 \cdot CT_1 \cdot (N \cdot O_{2,2} - RT_2 \cdot CT_2)^2 + RT_1 \cdot CT_2 \cdot (N \cdot O_{2,1} - RT_2 \cdot CT_1)^2}{N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2} + \frac{RT_2 \cdot CT_1 \cdot (N \cdot O_{1,2} - RT_1 \cdot CT_2)^2 + RT_2 \cdot CT_2 \cdot (N \cdot O_{1,1} - RT_1 \cdot CT_1)^2}{N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2}. \tag{3}$$

As a final step, we would need to compare whether $\chi^2 \geq t$. To do that, we calculate the numerator, and denominator of the fraction in Equation (3), separately. Subsequently, we multiply the denominator of the fraction with the threshold value $t$, and finally check inequality (4), without revealing any of the private inputs in the contingency tables.

$$\begin{aligned} RT_1 \cdot CT_1 \cdot (N \cdot O_{2,2} - RT_2 \cdot CT_2)^2 + RT_1 \cdot CT_2 \cdot (N \cdot O_{2,1} - RT_2 \cdot CT_1)^2 \\ + RT_2 \cdot CT_1 \cdot (N \cdot O_{1,2} - RT_1 \cdot CT_2)^2 + RT_2 \cdot CT_2 \cdot (N \cdot O_{1,1} - RT_1 \cdot CT_1)^2 \\ \stackrel{?}{\geq} t \cdot (N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2) \end{aligned} \tag{4}$$

### 3.1 Efficient Masking-Based Comparison

To the best of our knowledge the state-of-the-art techniques to perform secure comparisons, both in the homomorphic, and in the secret shared domain, require bitwise operations on the secret inputs. These operations have a high total cost, as their complexity is at best logarithmic in the number of bits per input. To allow for a practically efficient implementation of our solution, we consider a masking technique to perform the comparison instead of the bit-decomposition of our inputs. This masking technique works as follows: Consider $[\![x]\!]$ to be either the homomorphic encryption of $x$ or its secret shared value, where $x$ is assumed to be an integer. We check the inequality $[\![x]\!] \stackrel{?}{\geq} [\![y]\!]$ by computing $[\![\widehat{x-y}]\!] = [\![r]\!] \cdot [\![x-y]\!] + [\![r']\!]$, with $r$ and $r'$ random numbers satisfying the following condition: We have to choose $r$ and $r'$ such that the computation $[\![r]\!] \cdot [\![x-y]\!] + [\![r']\!]$ preserves the relation between $x$ and $y$. To do that we select $r$ to be a positive integer number (bounded properly so as to fit the largest possible input sizes we can handle), and $r' < r$ . Afterwards we reveal the masked value by respectively decoding or opening the calculated value. Depending on the sign of $r \cdot (\widehat{x-y}) + r'$ we can deduce the relationship between $x$ and $y$ (i.e., if $r \cdot (\widehat{x-y}) + r' > 0$ then $x > y$, otherwise $x < y$).

The proposed type of masking, which allows us to perform the comparison, could leak information about the secret input to the inequality, which in our case is the difference $x - y$. The leakage occurs if a party observing the masked result of the inequality check can submit multiple queries on the same inputs. This is possible because after several queries, if we select the maximum masked

value observed, it will be close to the bound set for the randomness $r$ times the difference $(x - y)$. Hence, by deduction, if we divide the maximum observed masked value by the upper bound on $r$, we will get a good approximation of the value $x - y$. To avoid such leakage, we require the random values $r$ and $r'$ to be selected once per query, and be thereafter fixed, until the contents of the query (i.e., the real inputs to the protocol) change. Note that in our setting the protocols will be only executed once every three months; only then may the input change. At this point we select new random values per query. This approach makes the aforementioned attack impossible in practice.

## 4 Homomorphic Encryption Approach

### 4.1 Setup and security assumptions

To solve the problem described in Section 3 with homomorphic encryption, we need multiple parties, as indicated in Figure 1. The steps of the process depicted in Figure 1 are as follows. In the first step, the decryptor will select the secret key, and associated public key for the homomorphic encryption, and make the public key available to all medical centers. Then, all the medical centers will encrypt their contingency tables with the given public key, and send these encryptions to the computation server. Upon receiving all contingency tables, the computation server will first add them to construct the aggregated contingency table, and subsequently perform the operations of the Pearson $\chi^2$ test. Then, the computation server will send the result, which is masked with the technique described in Section 3.1 to the decryptor, who uses the secret key to decrypt the masked value, and performs the comparison.

It is important to note that in this model we completely trust the decryptor to decrypt the result, and to post the corresponding value into the public table. To avoid this single point of trust, we can introduce a multiparty computation to perform the decryption based on a secret shared decryption key.

Other than that, the solution based on homomorphic encryption relies on the following two security assumptions:

- The computation server is honest but curious: It will follow the stated protocol to provide the desired functionality, and will not deviate, nor fail to return the results. The computation server can however monitor the result of every operation it performs.
- We only need the decryptor to perform the comparison, but he should not know anything about the input values. Therefore we presume that the communication between the centers, and the computation server is private. This can be achieved by performing the communication over authenticated, secure channels.

The honest but curious assumption for the computation server is reasonable to model an economically motivated cloud service provider. The cloud is motivated to provide excellent service, yet would take advantage of extra available information.

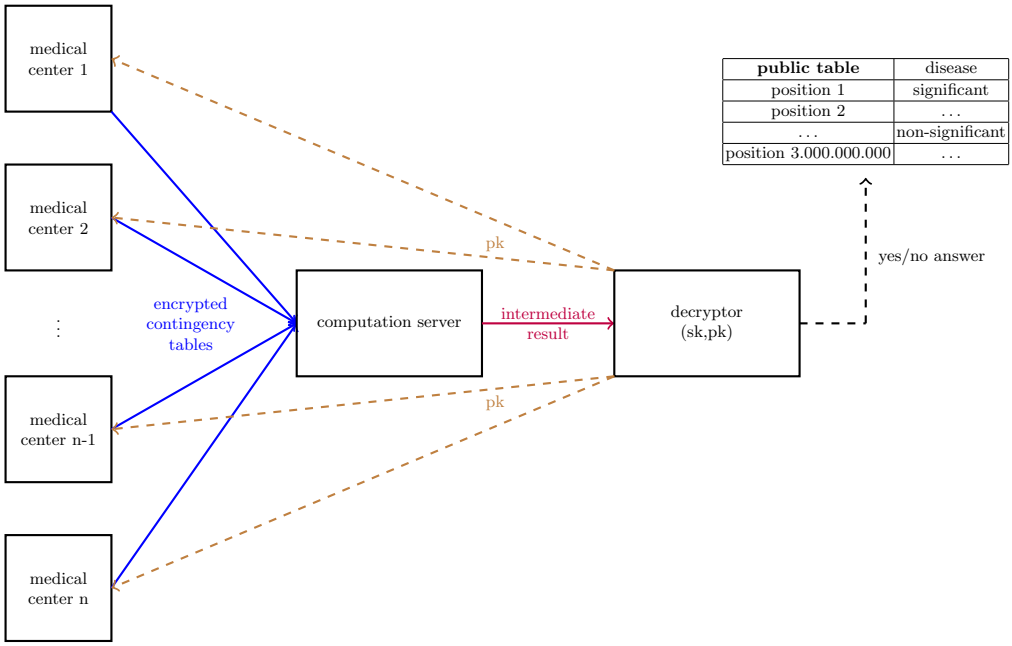


Fig. 1: A schematic representation of the homomorphic scenario.

For the homomorphic evaluation of the $\chi^2$ statistic we use the FV scheme, introduced by Fan and Vercauteren [FV12]. Moreover we base our implementation on the FV-NFLlib software library [Cry16] in which the FV homomorphic encryption scheme is implemented using the NFLlib software library developed for performing polynomial arithmetic computations (as described in [MBG+16], and released in [CIQ16]).

### 4.2 Preliminaries

**The Fan-Vercauteren SHE scheme** The Fan-Vercauteren SHE scheme is a scale invariant SHE scheme whose hardness is based on the ring learning with error problem (RLWE) [LPR13]. It operates on polynomials of the ring $R = \mathbb{Z}[X]/(f(X))$ with $f(X) = X^d + 1$ for $d = 2^n$.
The FV scheme makes use of a plaintext space $R_t$ with $R$ the polynomial ring defined above, and $t > 1$ a small integer modulus. Each coefficient of a plaintext polynomial is computed modulo $t$. The ciphertext space consists of a pair of elements of $R_q$ with $R$ the polynomial ring defined above, and $q > 1$, an integer modulus much larger than the plaintext modulus $t$. A homomorphic encryption scheme consists of a standard encryption scheme specifically constructed to enable additions, and multiplications in the ciphertext domain. The key generation, and encryption algorithms require elements sampled from two probability distributions defined on R. The secret key of our scheme is sampled from $\chi_{\text{key}}$, and for

encryption some error polynomials are sampled from an error distribution $\chi_{\text{err}}$. These probability distributions in combination with the degree $d$ of the defining polynomial $f$ of $R$, and the size of the integer $q$ determine the security of the FV scheme.
Given the parameters $d$, $q$, and $t$, and distributions $\chi_{\text{key}}$, and $\chi_{\text{err}}$, and using bold notation for a vector of two polynomials, we define the encryption, and decryption mechanism of the homomorphic encrytion scheme introduced by Fan and Vercauteren:

- Encrypt($\mathbf{pk}$,$m$): By multiplying the message $m \in R_t$ with $\Delta = \lfloor q/t \rfloor$ we transfer the message $m$ to the ring $R_q$. To hide the message we sample the error polynomials $e_1, e_2 \in \chi_{err}$, and $u \in \chi_{key}$, and compute the polynomials $c_0 = \Delta \cdot m + bu + e_1$, and $c_1 = au + e_2$. Both the polynomials $c_0$, and $c_1$ belong to $R_q$, and together they form the ciphertext $\mathbf{c} = (c_0, c_1)$ of the FV scheme.
- Decrypt(sk, $\mathbf{c}$): First compute $\tilde{m} = [c_0 + s \cdot c_1]_q$, then by scaling down the coefficients of $\tilde{m}$ by $\Delta$, and rounding the results we recover the message $m$.

Given a ciphertext $c = (c_1, c_2)$ we can write the $\tilde{m}$ of the decryption algorithm as $[c_0 + c_1 s]_q = \Delta \cdot m + e$, with $e$ the noise in the ciphertext. From this equation one can clearly see that if the noise $e$ grows too large, the decryption algorithm will fail to output the original message $m$ correctly. The property of ensuring that decryption results in the original message is called the correctness of the encryption scheme.
Every homomorphic operation will cause the noise in the ciphertexts to increase. Knowing the computations we want to perform in advance enables us to optimize the order of the computations for the sake of minimizing the noise growth. In addition it enables us to make an estimation of the noise present in the result, and hence allows us to determine parameters that can deal with this noise.

**Preprocessing of the data to improve the performance of the homomorphic computations** The plaintext space of the FV scheme consists of the polynomial ring $R_t$. Hence the first thing that needs to be done is encode the given integer data values into polynomials in the ring $R_t$. We achieve this by applying the $w$-NIBNAF encoding introduced in [BBB$^+$17]. The main idea of $w$-NIBNAF encoding goes back to Dowlin et al. [DGBL$^+$15] (see also [DGL$^+$16,LLAN15,NLV11]), and was analyzed in more detail by Costache et al. [CSVW16]. It consists of expanding the given number $\theta$ with respect to a base $b_w$, and replacing this base with the symbol $X$. So the encoding of a number $\theta$ is given by:

$$\theta = a_r X^r + a_{r-1} X^{r-1} + \cdots + a_1 X + a_0 - a_{-1} X^{d-1} - a_{-2} X^{d-2} - \cdots - a_{-s} X^{d-s} \quad .$$

For our encoding we use a well chosen non-integral base such that the encoding of an input value results in a polynomial with coefficients in the set $\{-1, 0, 1\}$, with the property that each set of $w$ consecutive coefficients has no more than one non-zero coefficient. Hence encoding our inputs with $w$-NIBNAF leads to

sparse polynomials.
Starting our computations with sparse polynomials with coefficients in the set $\{-1, 0, 1\}$ leads to small coefficients in the resulting polynomial. Therefore it enables us to work with a smaller plaintext modulus $t$, which improves the performance of the homomorphic encryption scheme. Bigger values for $w$ leads to longer, and sparser encodings, which reduce the minimum size for $t$ even further. However we have to ensure that when all the computations are done, decoding will still provide the correct answer. Therefore we need to carfully select our base $b_w$, which also determines the value of $w$.

**Selecting the optimal parameters** The three main concepts that will affect the selection of our parameters are:

- the security of the somewhat homomorphic FV scheme
- the correctness of the somewhat homomorphic FV scheme
- the correctness of the $w$-NIBNAF encoding

To take the security restrictions into account we rely on the work by Albrecht, Player, and Scott [APS15], and the open source LWE hardness estimator implemented by Albrecht [Alb04]. The latter estimates the hardness of the LWE problem based on three given parameters: the dimension $d$, the ciphertext modulus $q$, and a parameter $\alpha$, which is related to the error distribution $\chi_{err}$. It takes into account the currently known attacks on the learning with error problem.
The parameters that satisfy the restriction of the security implications of a security level of 90 bits are $q = 2^{186}$, $d = 4096$, $\sigma = 265$[7]. From the description of the FV scheme in Section 4.2, we know that for a ciphertext to be decrypted correctly the error cannot grow too much. Therefore we make an estimation of the infinity norm of the error in the ciphertext resulting from computing our circuit, and select parameters that keep the error small enough. This results in an upper bound for $t$. Since this upper bound is constructed specifically for our circuit it will depend amongst other things on the number of centers that deliver data to the computation server.
For correctness of the $w$-NIBNAF decoding it is important to estimate the size of the coefficients of the encodings after performing the necessary operations. This leads to a lower bound on the plaintext modulus $t$. The security, and correctness of the FV scheme on one hand, and the correctness of the $w$-NIBNAF encoding on the other hand, set conflicting requirements for $t$. In order to solve this conflict we make use of the Chinese Remainder Theorem to decompose the plaintext space, following the idea of [BLLN13, §5.5]. This implies that we choose $t$ to be the product of small prime numbers $t_1, t_2, \ldots, t_n$, with $\forall i \in \{1, \ldots, n\} : t_i \leq t_{\max}$, and $t = \prod_{i=1}^{n} t_i \geq t_{\min}$, where $t_{\max}$ is determined by the security, and correctness of the FV scheme, and $t_{\min}$ by the correctness of the $w-$NIBNAF decoding.

[7] $\sigma$ determines the error distribution $\chi_{err}$.

### 4.3 Privacy-Preserving Homomorphic Chi-Squared Thresholding Algorithm and Parameters

In this section we combine all information given before in order to construct the algorithm needed to perform the privacy-preserving $\chi^2$ thresholding in the homomorphic setting. Algorithm 1 lists the computations for the algorithm corresponding to the order in which they need to be performed, and mentions the parties that need to perform them. Hence the steps from Figure 1 are described in more detail in Algorithm 1, and the combination of these two gives a clear picture of our homomorphic solution for the privacy-preserving $\chi^2$ thresholding.

**Algorithm 1** Privacy-preserving Homomorphic Chi-squaredTest

**Medical center:**

input: $O[2][2]$ = the observed values of the $2 \times 2$ contingency table (mutation vs. disease)

**for** $C_k$ medical center $k$ **do**

1. Encode each value $O^k_{i,j}$ for $i = 1, 2$; $j = 1, 2$ using the $w$-NIBNAF encoding. technique
2. Transform this encoding to the plaintext ring $R_{t_i}$ for all $i$ such that $t = \prod_i t_i$.
3. Encrypt the plaintext polynomials using the Fan-Vercauteren SHE scheme to obtain ciphertexts $\mathbf{c^k_{i,j}}$.

**end for**

**Computation center:**

input: encrypted contigency table values $\mathbf{c^k_{i,j}}$, for $k = 1, ..., Nc$, with $Nc$ the number of centers contributing data

4. Compute the four values of the aggregated contingency table $\mathbf{c_{m,n}} = \sum_{k=1}^{Nc} c^k_{m,n}$ .
5. Compute the $\chi^2$ numerator $\alpha$ and denominator $\beta$ homomorphically
6. Compute the difference of the $\chi^2$ numerator and the product of the encrypted $\chi^2$ threshold $T$ and denominator, and mask the computed difference with the random values $r$ and $r'$: $MR = r \cdot (\alpha - T \cdot \beta) + r'$

**Decryptor:**

input: $MR$ the encrypted masked output value of the homomorphic circuit

7. Decrypt the masked result $MR$
8. Use the inverse CRT ring isomorphism to transfer the plaintext polynomials to the ring $R_t$.
9. Decode the $w$-NIBNAF polynomial, and evaluate the result in the correct basis $b_w$ to get the value $MaskedDifference$ of the masked result.
10. Determine the significance based on the sign of $MaskedDifference$:

**if** $MaskedDifference > 0$ **then**

**return** 1

**else**

**return** 0

**end if**

For the homomorphic solution we consider two scenarios: (1) the case in which we compute the numerator and denominator of the $\chi^2$ statistic seperately, and

decrypt them both, and (2) the case where we use the masking technique, and decode the masked value to determine the "True/False" answer to the significance question. As mentioned before there are many parameters we need to set, and their values depend highly on the homomorphic circuit we perform, and the values of other parameters. We selected parameters for different scenarios in which we fix the number of patients per center to 10000 and vary the number of centers from 20 to 100. The parameters are listed in Table 2 and 3. We use these paramters in our implementation later to assess the performance by measuring the computation time, and communication cost for each of the three parties mentioned in Algorithm 1.
The value of $w$ and the splitting degree are dependent on the size of the numbers we need to encode, but independent of the number of centers included in the calculation. Therefore the value of $w$ and the splitting degree will be the same for all scenarios. Without comparison, they are respectively $w = 271$, and splitting degree = 3166. The parameters that differ for different number of centers in scenario (1) is the value for $t$, and its CRT factors. These are listed in Table 2. For scenario (2) in which we perform the masked comparison, the value of $w$ will be different for each different number of centers participating in the computations, because we have to encode the random variables $r$ and $r'$, which have sizes depending on the number of centers included in the computations. The parameters for the second scenario are listed in Table 3.

Table 2: Parameter selection for scenario (1)

| **Centers** | **Patients** | **t** |
|---|---|---|
| 20 | 200000 | $17431 \cdot 17443 \cdot 17449$ |
| 40 | 400000 | $1718713 \cdot 1718719 \cdot 1718723$ |
| 60 | 600000 | $1558103 \cdot 1558129 \cdot 1558177$ |
| 80 | 800000 | $1453043 \cdot 1453057 \cdot 1453061$ |
| 100 | 1000000 | $1376213 \cdot 1376231 \cdot 1376237$ |

Table 3: Parameter selection for scenario (2)

| **Centers** | **Patients** | **bit-length random value** | **w** | **splitting degree** | **t** |
|---|---|---|---|---|---|
| 20 | 200000 | 100 | 118 | 3625 | $12253 \cdot 12263 \cdot 12269$ |
| 40 | 400000 | 106 | 113 | 3629 | $885127 \cdot 885133 \cdot 885161$ |
| 60 | 600000 | 110 | 111 | 3644 | $802651 \cdot 802661 \cdot 802667$ |
| 80 | 800000 | 112 | 110 | 3653 | $747811 \cdot 747827 \cdot 747829$ |
| 100 | 100000 | 114 | 108 | 3651 | $707801 \cdot 70813 \cdot 707827$ |

### 4.4 Implementation and performance analysis

In order to assess the practical performance, and verify the correctness of the selected parameters of the homomorphic scenario, we implemented the privacy-preserving $\chi^2$ computation using the FV-NFLlib software library [Cry16]. Our presented timings are obtained by running the implementation on a computer equipped with an Intel Core i5-4590 CPU, runnning at 3.30 GHz. We executed the program 10 times per case, and calculated the average execution time for our timing results. To evaluate the scalability of our protocol we have considered the cases where our system receives data from 20, 40, 60, 80, and 100 medical centers, respectively. We assume each medical center to contribute data of 10000 subjects (i.e., the total number of subjects per case is 200000, 400000, 600000, 800000, and 1000000, respectively). In order to measure these timings we used the parameter set corresponding to the same scenario, so Table 2 for scenario (1) and Table 3 for scenario (2).

For scenario (1) we compute the numerator and denominator seperately, and do not perform the masked comparison with the threshold value. The CPU time needed for the hospitals to encrypt the four values of the contingency table is the same for any number of centers considered in the experiment. For our selected parameters this is 15.2 ms. Also the time to decrypt the numerator and denominator of the $\chi^2$ value is the same for any number of centers contributing to the experiment. The average time we measured during our experiments is 38.8 ms. The timings of the computation server in scenario (1) are the times needed to perform the calculations for computing the numerator and denominator of the $\chi^2$ statistic. These timings are dependent on the number of centers that participate in the computation. Therefore we list them in Table 4. We see that the timings for increasing centers do not differ significantly. This is consitent with the fact that homomorphic additions are not the most time consuming part of our computations.

For scenario (2) the encryption time does not depend on the number of centers either, since the centers can perform the encryption in parallel. The measured encryption time for one contingency table in scenario (2) is 17.1 ms. The time to decrypt the result does not depend on the number of centers participating in the computation either. However it is smaller than for scenario (1), since now we only have to decrypt one value instead of two. The measured decryption time for scenario (2) is 21.1 ms. The timings for the computation server are listed in Table 4, since these timings are dependent of the number of medical centers participating. Here we see the same trend as for scenario (1): the timings do not increase linearly in the number of medical centers.

From Table 4 one sees that the timings for the computation server do not differ much between both scenarios. This is because addition and multiplication with a constant (which are the extra operations we need to compute the masked value) are not the most time consuming homomorphic operations. Hence our

masked comparison gives an efficient solution for keeping the $\chi^2$ value private. We can also conclude that considering CPU time, our solution scales really well for increasing number of medical centers participating in the computation.

Table 4: CPU time computation server

| Centers | Patients | scenario (1) | scenario (2) |
|---|---|---|---|
| 20 | 200000 | 1.40 ms | 1.48 ms |
| 40 | 400000 | 1.48 ms | 1.52 ms |
| 60 | 600000 | 1.44 ms | 1.53 ms |
| 80 | 800000 | 1.47 ms | 1.56 ms |
| 100 | 100000 | 1.49 ms | 1.56 ms |

For the homomorphic setup, there is no communication cost during the computations. The communication cost comes from sending values from each of the three parties to the next. We have three points of communication: the public key has to be sent from the decryptor to the medical centers; the encrypted values of the contingency tables have to be sent from the medical centers to the computation server; and the result has to be sent from the computation server to the decryptor. The communication cost is similar for both scenarios since for both scenarios the size of one ciphertext will be the same, and we only need to sent ciphertexts form one party to another. The size of the public key that needs to be sent to the different medical centers is 186 kB. The data needed to send one contingency table to the computation server is 2.1 MB. The communication cost between the medical centers, and the computation server is the number of centers participating times the number of data needed to send one contingency table. So this communication cost increases linearly in the number of centers contributing to the computation. In scenario (1) we send both the numerator as the denominator from the computation server to the decryptor, which results in a communication cost of 1.8 MB. In scenario (2) we only have to send one value, which gives a communication cost of 0.54 MB.

## 5 Secure Multiparty Computation Approach

To address the challenge of disease gene identification using secure multiparty computation techniques, in the setting described in Section 3, we deploy MASCOT [KOS16]. We selected MASCOT [KOS16] as the most suitable multiparty computation solution, because it is currently the most efficient proposal, offering malicious static security with a dishonest majority. This means that any number of the computing parties may deviate from the protocol execution, and this will be detected without leaking information, other than what the correct protocol

execution would reveal. Corruption may only occur prior to the beginning of the protocol execution, affecting up to $n-1$ (out of the $n$) computing parties.

### 5.1 Setup and Security Assumptions

For our multiparty computation approach, we first need to determine the number of computation servers $n$ ($n \geq 2$) that we have at our disposal. Given that the underlying protocol offers security against any coalition of $n-1$ computation servers, we consider the security of the whole system to increase as the number of computation servers increases. However, the number of computation servers is inversly proportional to the efficiency of the solution. Therefore, we consider that three computation servers is an adequate number of servers, both from an efficiency/plausibility perspective, and from a security perspective. If any two of the three computation servers that we assume get compromised, or otherwise behave dishonestly, or even collude, the solution still guarantees input privacy, and does not accept incorrect results.

We assume a preprocessing phase that can take place offline, at any moment prior to the actual protocol execution. This is to create the necessary randomness for the medical centers to contribute their inputs in a secret shared manner to the computation servers. In addition, the preprocessing phase creates authenticated randomness to be used in the online phase, so as to boost the efficiency of computing multiplications on the shares, which requires interaction amongst the servers.

The medical centers that wish to contribute their private inputs, first need to agree on a common format for this data (e.g., what is the order of sending the contingency tables). Then, they need to secret share their contingency tables to the three computation servers, which can also be pushed to an offline, preprocessing phase. Given that all contributing medical centers have shared their private contingency tables to the computation servers, the online phase starts. During the online phase the servers perform both local, and interactive secure computations, and they finally reveal per contingency table whether the relationship between a mutation at a certain DNA position, and a disease is significant or not, without disclosing further information on the underlying data. A schematic representation of this approach is presented in Figure 2.

### 5.2 Preliminaries

**Additive Secret Sharing** A secret sharing scheme is a protocol, which allows (some of) the protocol participants to share their secret inputs amongst all other protocol participants, in such a way that nothing is revealed to the individual participants about the secret input. In some instances of secret sharing schemes, a *subset* of the protocol participants, called the qualified set, can reconstruct the original secret input, when they engage in the reconstruction protocol. Other schemes, such as additive secret sharing, require *all* protocol participants to contribute their shares for the reconstruction protocol to work.

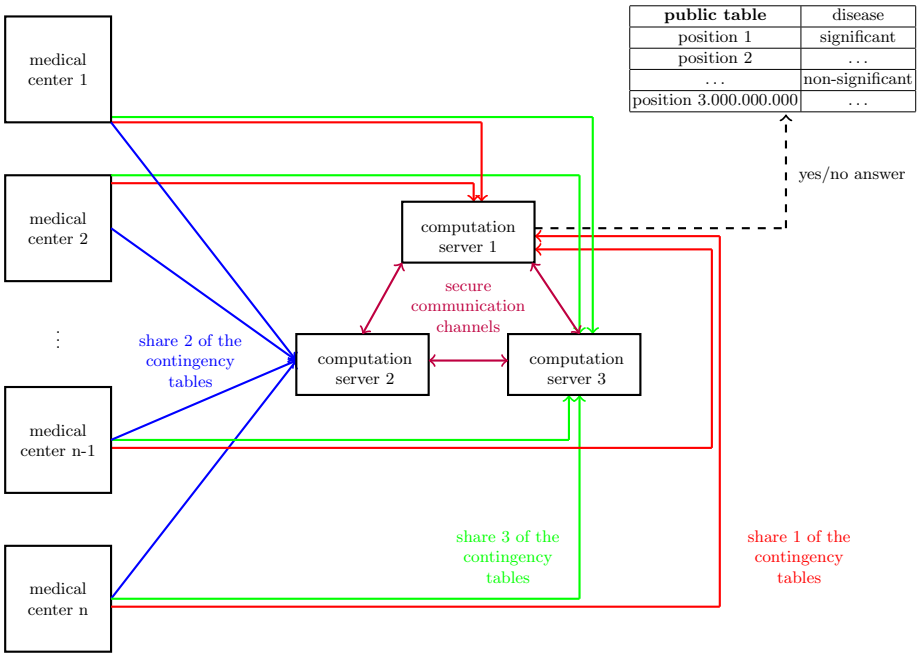


Fig. 2: A schematic representation of the multiparty computation scenario.

Additive secret sharing is essentially masking the input of a protocol participant $x$ by subtracting a random value $r$ from it. Given that the value of $r$ is only known to the inputting participant, and that the rest of the participants hold shares $[\![r]\!]$ of this value (which can be done in a preprocessing stage), secure shares of $x$ can be created as follows: The inputting party computes $\epsilon = x - r$, and broadcasts it to the rest of the participants. All other parties can now compute their own shares of the input $x$ as $[\![x]\!] = [\![r]\!] + \epsilon$ . It is easy to see that this scheme enjoys an additively homomorphic property, allowing additions of shares, and linear functions to be directly computed locally on the shares. Hence, no communication amongst the protocol participants is required to perform additions on these additively secret shared values.

**Oblivious Transfer** Oblivious Transfer (OT) is a cryptographic primitive, which allows a sender to transfer only one (or none) out of many values to a receiver, while remaining oblivious as to which value has been received (if any). Oblivious transfer was introduced by Rabin [Rab81] in 1981. Basic OT constructions make use of public key encryption primitives to allow the aforementioned functionality. More precisely, an 1-out-of-2 OT requires the sender to be in possession of a public/private key pair, generate, and send two random messages to the receiver, decrypt two messages, and send two messages to the receiver. The receiver obtains the public key of the sender, along with the two

random messages that the latter has selected, performs one encryption, and one blinding, sends the resulting message to the sender, and finally inverts the blinding to retrieve the desired message. For more information on basic OT, we refer the reader to the work of Even et al. [EGL85].

Many works have considered extending OT so as to make it practically efficient, with the most notable work of Ishai et al. [IKNP03]. Under the assumption that a random oracle $H$, which can be instantiated by a hash function family, exists, Ishai et al. [IKNP03] show how to perform only a few OTs from scratch, and then be able to perform many additional OTs at the cost of a constant number of invocations of the random oracle. The actively secure version of their protocol comes with an increase at the cost of the protocol by a factor $\sigma$, for $\sigma$ a statistical security parameter. More recently, Keller et al. [KOS15] presented an actively secure OT extension protocol, where malicious security comes at negligible extra cost.

**Correlated Oblivious Product Evaluation - COPE** The Correlated Oblivious Product Evaluation (COPE) protocol presented by Keller et al. [KOS16] is essentially a generalization of Ishai et al.'s protocol [IKNP03] to the arithmetic case (instead of the original binary). The protocol is executed between two parties, and allows them to obtain an additive sharing of the product $x \cdot \Delta$, where the sender holds $x \in \mathbb{F}$, and the receiver holds $\Delta \in \mathbb{F}$. COPE is based on Gilboa's oblivious product evaluation [Gil99], where the parties run $k$ sets of 1-out-of-2 OTs, on $k$-bit inputs. The proposed product evaluation is correlated in the sense that the one party's input $\Delta$ is fixed at the beginning of the protocol for many protocol runs. After a one-time expensive initialization of the COPE protocol, the extension step, generating fresh OTs without the public-key crypto extensive costs, can be repeated several times on new inputs $x$.

**MASCOT Online Phase** The online phase of MASCOT [KOS16] is essentially the same as the one of the SPDZ protocol [DPSZ12,DKL$^+$13]. This family of protocols uses additive secret sharing, allowing additions, and linear functions to be computed locally by the protocol participants, without requiring communication. To achieve active security, information-theoretic MACs are being used, which provide authenticity, and integrity of the messages. A secret shared value $x$, shared amongst $n$ parties, is represented as follows:

$$[\![x]\!] = (x^{(1)}, ..., x^{(n)}, m^{(1)}, ..., m^{(n)}, \Delta^{(1)}, ..., \Delta^{(n)}), \tag{5}$$

where, $x^{(i)}$ is the random share, $m^{(i)}$ is the random MAC share, and $\Delta^{(i)}$ is the MAC key share, such that $m = x \cdot \Delta$.

To perform multiplications of secret shared values, multiplication triples à la Beaver [Bea91] from the preprocessing phase are required, with which the parties can compute shares of the required product. The multiplication triples are of the form: $([\![a]\!], [\![b]\!], [\![c]\!])$, with $a$, $b$ uniformly random, and $c = a \cdot b$. More precisely, to compute $[\![z]\!] = [\![x \cdot y]\!]$, on input $[\![x]\!]$, $[\![y]\!]$, and given a multiplication

triple $([\![a]\!], [\![b]\!], [\![c]\!])$, the parties compute $\epsilon = [\![x]\!] - [\![a]\!]$, and $\rho = [\![y]\!] - [\![b]\!]$, and open these two values. Then, they compute $[\![z]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$ .

**MASCOT Offline Phase - Preprocessing** The goal of the preprocessing phase is to generate random values for the parties who wish to contribute inputs –so as to allow them to mask their inputs in an authenticated manner–, and multiplication triples –so that multiplication of secret shared values can be efficiently implemented in the online phase–.

The preprocessing is based on Oblivious Transfer techniques, and more precisely the authors of MASCOT [KOS16], have generalized the OT extension idea presented by Ishai et al. [IKNP03] to the arithmetic circuit case. For every party that wishes to contribute an input, the COPE protocol is executed, to create an authenticated version of the input, based on the global MAC key $\Delta$. Creation of (authenticated) additive shares is straightforward. Recall that both the shares, and their MACs are linear. Hence, computing linear functions on (authenticated) shared values is also straightforward. To ensure active security, the party that wishes to contribute an input first authenticates a random input $x_0$ together with the actual inputs $m$. Then, the party opens a random linear combination of the inputs including $x_0$, and all other parties check the MAC on this linear combination. This way the party contributing input is committed to them, and the actual inputs are masked by the random input $x_0$.

For the triple generation, the parties invoke an OT to compute the secret sharing of $b \in \mathbb{F}$, and $\mathbf{a} \in \mathbb{F}^\tau$, where $\tau \geq 3$ is a security related parameter, meaning that they run $\tau$ copies of the basic two-party COPE per pair of parties. This ensures that $\mathbf{a}$ has enough randomness to produce a triple. To protect against malicious behavior, the parties sample two sets of random coefficients $\mathbf{r}$, and $\hat{\mathbf{r}} \in \mathbb{F}^\tau$, from which they generate two triples with the same $b$ component. Upon authentication of their shares, the parties ensure correctness of one of the triples, by sacrificing the other triple.

### 5.3 Privacy-Preserving Chi-Squared Thresholding Protocol

Our protocol calculates the units of inequality (4) using MASCOT [KOS16]. For the inequality check, we apply the masking technique described in Section 3.1. In addition to the masking-based comparison, we have also implemented the protocol using the standard, bit-decomposition based secure comparison, as implemented in SPDZ-2 [Bri16]. We have also made a non-secure implementation avoiding completely the comparison. All these cases are presented in Section 5.4, analyzing their performance. The online phase of the masking-based version of our protocol is detailed in Algorithm 2.

In our setting, we consider the computing parties that actually execute our protocol (i.e., the computation servers), different from the parties contributing their inputs (i.e., the medical centers), as shown in Figure 2. For the offline phase, together with the preparation of the triples, and randomness discussed in Section 5.2, we wish to perform the required preprocessing that will allow

**Algorithm 2** $v \leftarrow \mathsf{Chi\text{-}squaredTest}(Nc, \mathbf{N}[Nc], [\![\mathbf{O}[4][Nc]]\!], \chi^2, [\![r]\!], [\![r']\!])$

**Input:** $Nc$ = number of centers contributing data,
$\mathbf{N}[Nc]$ = a table of size $Nc$ containing the total sample size $N_i$ of every center $i$,
$[\![\mathbf{O}[4][Nc]]\!]$ = secret shared *observed* values of the $2 \times 2$ contingency table (mutation vs. disease), contributed by each of the $Nc$ centers,
$\chi^2 = \chi^2$ threshold value for the significance test,
$[\![r]\!], [\![r']\!]$ = secret shared random values $r$ and $r'$
**Output:** $v = 0$ or 1; $0 \rightarrow$ non-significant relationship between mutation, and disease, $1 \rightarrow$ significant relationship between mutation, and disease
**for all** $P_j$ **do**
  {*Each party* $P_j$ *engages in the protocol*}
  **for all** $C_i$ **do**
    $[\![O_{k,l}]\!] \leftarrow [\![\sum_{i=1}^{Nc} O_{k,l}]\!]_i, k = 1, 2; l = 1, 2$
  **end for**
  $N \leftarrow \sum_{i=1}^{Nc} N_i$
  $[\![RT_i]\!] \leftarrow [\![O_{i,1} + O_{i,2}]\!], i = 1, 2$
  $[\![CT_i]\!] \leftarrow [\![O_{1,i} + O_{2,i}]\!], i = 1, 2$
  $[\![model_{k,l}]\!] \leftarrow [\![RT_k \cdot CT_l]\!], k = 1, 2, l = 1, 2$
  $[\![square]\!] \leftarrow [\![(N \cdot O_{1,1} - model_{1,1})^2]\!]$
  $[\![U_{i,j}]\!] \leftarrow [\![square \cdot model_{i,j}]\!], i = 1, 2; j = 1, 2$
  $[\![numerator]\!] \leftarrow [\![\sum_{i=1,j=1}^{2,2} U_{i,j}]\!]$
  $[\![denominator]\!] \leftarrow N \cdot [\![model_{1,1} \cdot model_{2,2}]\!]$
  $[\![difference]\!] \leftarrow [\![numerator]\!] - \chi^2 \cdot [\![denominator]\!]$
  $[\![MaskedDifference]\!] \leftarrow [\![difference \cdot r + r']\!]$
  $MaskedDifference \leftarrow \mathsf{Open}([\![MaskedDifference]\!])$
  **if** $MaskedDifference > 0$ **then**
    **return** 1
  **else**
    **return** 0
  **end if**
**end for**

the medical centers to correctly contribute their inputs, without compromising privacy. To do that we use the protocols proposed by Damgård et al. [DDN+15]. First we use the Output Delivery protocol to reveal a preprocessed random value $r$ only to the inputting party, who can then broadcast his masked input $x - r$ to the computing parties. Based on this value, and the preprocessed randomness $r$, the servers can locally compute their share of $x$, as $[\![x]\!] = (x - r) + [\![r]\!]$.

### 5.4 Implementation and Performance Analysis

We have built a proof of concept implementation of our MPC approach using the platform provided by the authors of MASCOT [KOS16] in SPDZ-2 [Bri16]. We ran our experiments for timing the execution of our protocol on a desktop computer equipped with an Intel(R) Core(TM) i5-3570K processor, at 3.40GHz, with 16.00 GB RAM, and the Ubuntu 17.04 operating system.

We have only considered the online phase of the protocol, as the preprocessing is protocol-independent, and can be executed at any moment, well before the execution of the online phase. We note, however, that the offline phase is also practically efficient, and we refer the reader to MASCOT [KOS16] for more details on the throughput of the offline phase. To give an indication of the cost of the offline phase of the protocol, we estimate the triple generation throughput, based on the experiment results presented by Keller et al. [KOS16]. For three computation servers, equipped with eight-core i7 3.1 GHz CPU, and 32 GB RAM, in a local network with a 1 Gbit/s link per party, and a field $\mathbb{F}_p$, with $p$ a 128-bit prime, approximately 2200 triples per second can be generated. Every time we recorded timings, before the execution of the online phase, we ran the setup script provided with the SPDZ-2 [Bri16] software. This script simulates the offline phase, and creates all the necessary randomness for the execution of the online phase. The fact that the offline phase is simulated does not affect the performance, or efficiency of the online phase.

Our experiments were conducted on localhost with three computation servers. Hence, we do not take the network latency into account in the timing results we report. We do present the size of the data that each server has to send, as well as the communication rounds, and we consider this information to be sufficient for the reader to calculate the additional communication cost, based on the available network bandwidth.

For our performance analysis we have considered the following three scenarios: (1) the case where we calculate the numerator, and denominator of the $\chi 2$ statistic in the secret shared domain, and then open these two results; (2) the case where the secure comparison is implemented as described in Section 3; and (3) the case where we perform the secure comparison in the secret shared domain, and then open only the "True/False" answer to this question, as implemented in MASCOT. We selected these three scenarios, as the secure comparison is the most costly operation we need to carry out, and we wish to assess its impact on the performance of our protocol. Thus, with scenario (1) we completely avoid the secure comparison by opening the numerator, and denominator separately; with scenario (2) we perform the secure comparison using our randomization

approach; and with scenario (3) we use the most popular method to carry out a secure comparison (see [CDH10]), which is based on bit-decomposition –an inherrently inefficient approach–. Note that scenario (1) does not satisfy the security requirements of our application, and is presented only for the sake of performance comparison.

For all our timing results we have executed our protocol 10 times per case, and calculated the average execution time. The communication cost of the protocol is constant. For our experiments we have established that all input data is shared by one of the computation servers (namely Server 1), instead of the medical centers that would contribute the data in a real setting. This is reflected in the communication cost of the protocol for Server 1, which has to secret share all input data. Note, that in practice the secret sharing step can be pushed to an offline preprocessing phase.

In Table 5 we present the execution times of our approach, as well as the data sent by Server 1 (including the sharing of the original inputs), without performing a secure comparison (scenario 1). Server 1 is presented separately, because it has to do some extra tasks, such as sharing the inputs, collect all the final results, and print them, which is reflected in its execution times. The other two servers are grouped together, as their execution times are similar. Although we would expect the execution times to grow with the number of medical centers contributing data, this is not the case. This is because the execution times are so small that they can be highly affected by the computing environment. Furthermore, the communication cost of Server 1 includes also the secret sharing of the inputs. This is how all three scenarios have been executed. Recall, however, that the sharing of the inputs can be performed in a preprocessing phase, prior to the actual protocol execution, allowing the online phase to be less communication intensive. The communication cost for the other two servers is constant –1228 bytes–, since they do not share any inputs. The protocol completes its execution in 4 communication rounds, and consumes 9 triples, and 1 square from the preprocessing.

Table 5: Performance with No Secure Comparison

| | | **Server 1** | | **Server** $i$, $i \neq 1$ |
|---|---|---|---|---|
| **Centers** | **Patients** | **CPU Time** | **Sent Data** | **CPU Time** |
| 20 | 200000 | 1.6 ms | 4152 bytes | 1.3 ms |
| 40 | 400000 | 1.6 ms | 6712 bytes | 1.4 ms |
| 60 | 600000 | 1.5 ms | 9272 bytes | 1.3 ms |
| 80 | 800000 | 1.7 ms | 11832 bytes | 1.4 ms |
| 100 | 1000000 | 1.7 ms | 14392 bytes | 1.4 ms |

In Table 6 we provide the execution times of our alternative secure approach, which is based on randomizing the difference of the two numbers to be compared (scenario 2). Note that in the previously presented scenario (1) it is sufficient

to work in a prime field of 128 bits, as the numbers we operate with never grow larger than 114 bits. To perform the randomization of the difference of our numbers securely, however, we need to multiply them with a (random) number of roughly the same size. This implies that we need to work in a field of 256 bits to be able to handle the size of our quantities. More precisely, we calculated the largest that our quantities can grow in all cases, and we selected the random numbers to be upper bounded by these sizes. Our numbers can grow up to 100, 106, 110, 112, and 114 bits for 20, 40, 60, 80, and 100 medical centers, respectively. For this scenario, and the case of 20 centers, Server 1 has to send 82 (256-bit) elements (instead of 128-bit elements) to the two other servers. The two additional elements that have to be sent are the randomness $r$, and $r'$, which is used to mask the difference that facilitates the execution of the secure comparison. The communication cost for Server 1 is analyzed in Table 6, while for the other two servers is constant and equal to 1632 bytes. The protocol completes its execution in 5 communication rounds, and consumes 10 triples, and 1 square from the preprocessing.

Table 6: Performance with Randomized Secure Comparison

| | | **Server 1** | | **Server *i*, $i \neq 1$** |
|---|---|---|---|---|
| **Centers** | **Patients** | **CPU Time** | **Data Sent** | **CPU Time** |
| 20 | 200000 | 1.7 ms | 7616 bytes | 1.4 ms |
| 40 | 400000 | 1.7 ms | 12736 bytes | 1.5 ms |
| 60 | 600000 | 1.9 ms | 17856 bytes | 1.7 ms |
| 80 | 800000 | 2.1 ms | 22976 bytes | 1.8 ms |
| 100 | 1000000 | 2.2 ms | 28096 bytes | 1.9 ms |

In Table 7 we display the execution times of our protocol, using the standard, bit-decomposition based, secure comparison (scenario 3). Similarly to the second scenario, due to the size of our inputs (up to 114 bits), we need to scale our inputs representation to 256 bits field elements, so as to achieve adequate statistical security (always $\geq$ 40 bits). The protocol communication cost of Server 2, and 3 is constant –4244 bytes–, while for Server 1 it varies, based on the number of inputs it has to share, as shown in Table 7. The protocol completes its execution in 10 communication rounds, and consumes 50 triples, 1 square, and 31 bits from the preprocessing. As expected, this is the most inefficient of the three scenarios, both in terms of communication, and in terms of computational cost.

## 6 Conclusion

Our work shows that full-scale privacy-preserving GWAS deployment has become practical. Our solutions provide provable security guarantees, while being efficient for realistic sample sizes, and number of medical centers contributing

Table 7: Performance with Standard Secure Comparison

| | | Server 1 | | Server $i$, $i \neq 1$ |
|---|---|---|---|---|
| **Centers** | **Patients** | **CPU Time** | **Data Sent** | **CPU Time** |
| 20 | 200000 | 2.2 ms | 12712 bytes | 1.9 ms |
| 40 | 400000 | 2.3 ms | 17832 bytes | 2.0 ms |
| 60 | 600000 | 2.3 ms | 22952 bytes | 2.0 ms |
| 80 | 800000 | 2.5 ms | 28072 bytes | 2.2 ms |
| 100 | 1000000 | 2.4 ms | 33192 bytes | 2.1 ms |

data to the studies. Moreover, our solutions scale logarithmically in the number of subjects contributing data to the study, which means that as GWAS population sizes grow, our approach will remain suitable. We also propose a new masking-based comparison method, and show that in certain application scenarios, such as the GWAS scenario at hand, comparisons can be executed efficiently, without leaking information about the underlying data.

From the setup description of both suggested techniques, one can determine the first significant difference: in the homomorphic setting, the medical centers only have to encrypt, and share their data with one party, namely the computation server; while for the multiparty computation they have to share their data with two or more computation servers. The execution times resulting from our experiments show that the MPC approach is significantly faster than the homomorphic approach. Even if we assume the encryption of the contingency tables by the medical centers to be part of a preprocessing phase, the homomorphic approach will take more than a second to complete its execution, while the computations in the MPC setup take only a few milliseconds. In terms of communication cost, the homomorphic setup has the advantage that it needs no communication during the computations. However, in terms of total amount of data that has to be transferred between the different parties, the MPC setup outperforms the homomorphic setup once more. We therefore recommend the MPC approach, as it is the most efficient out of the two approaches, and it does not rely on the strong assumption of semi-honest parties participating in the protocol.

## References


[Alb04] Martin Albrecht. Complexity Estimates for Solving LWE. `https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py`, 2000–2004.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the Concrete Hardness of Learning with Errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.

[ASD+14] Amin Ardeshirdavani, Erika Souche, Luc Dehaspe, Jeroen Van Houdt, Joris Robert Vermeesch, and Yves Moreau. NGS-Logistics: Federated Analysis of NGS Sequence Variants Across Multiple Locations. *Genome medicine*, 6(9):71, 2014.

[BBB+17] Charlotte Bonte, Carl Bootland, Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Faster Homomorphic Function Evaluation using Non-Integral Base Encoding. Cryptology ePrint Archive, Report 2017/333, 2017. http://eprint.iacr.org/2017/333.

[Bea91] Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Martijn Stam, editor, *Cryptography and Coding 2013*, volume 8308 of *LNCS*, pages 45–64. Springer, 2013.

[Bri16] Bristol Crypto. SPDZ-2: Multiparty Computation with SPDZ Online Phase and MASCOT Offline Phase. https://github.com/bristolcrypto/SPDZ-2, 2016.

[CDH10] Octavian Catrina and Sebastiaan De Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.

[CIQ16] CryptoExperts, INP ENSEEIHT, and Quarkslab. NFLlib. https://github.com/quarkslab/NFLlib, 2016.

[Cry16] CryptoExperts. FV-NFLlib. https://github.com/CryptoExperts/FV-NFLlib, 2016.

[CSVW16] Anamaria Costache, Nigel P Smart, Srinivas Vivek, and Adrian Waller. Fixed Point Arithmetic in SHE Schemes. In *SAC 2016*, LNCS. Springer, 2016.

[CTW+15] Scott D Constable, Yuzhe Tang, Shuang Wang, Xiaoqian Jiang, and Steve Chapin. Privacy-Preserving GWAS Analysis on Federated Genomic Datasets. *BMC medical informatics and decision making*, 15(5):S2, 2015.

[DDN+15] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential Benchmarking based on Multiparty Computation. *IACR Cryptology ePrint Archive*, 2015:1006, 2015.

[DGBL+15] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Manual for Using Homomorphic Encryption for Bioinformatics. Technical report, November 2015.

[DGL+16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *International Conference on Machine Learning*, volume 48, pages 201–210. JMLR.org, 2016.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical Covertly Secure MPC for Dishonest Majority–or: Breaking the SPDZ Limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[fGH16] The Global Alliance for Genomics and Health. A Federated Ecosystem for Sharing Genomic, Clinical Data. *Science*, 352(6291):1278–1280, jun 2016.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[Gil99] Niv Gilboa. Two Party RSA Key Generation. In *Annual International Cryptology Conference*, pages 116–129. Springer, 1999.

[HSR+08] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures using High-Density SNP Genotyping Microarrays. *PLoS genetics*, 4(8):e1000167, 2008.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

[KBLV13] Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. A New Way to Protect Privacy in Large-Scale Genome-Wide Association Studies. *Bioinformatics*, 29(7):886–893, 2013.

[KL15] Miran Kim and Kristin Lauter. Private Genome Analysis through Homomorphic Encryption. 15:S3, 12 2015.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively Secure OT Extension with Optimal Overhead. In *Annual Cryptology Conference*, pages 724–741. Springer, 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.

[LLAN14] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private Computation on Encrypted Genomic Data. In *International Conference on Cryptology and Information Security in Latin America*, pages 3–27. Springer, 2014.

[LLAN15] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. *Private Computation on Encrypted Genomic Data*, pages 3–27. Springer International Publishing, Cham, 2015.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60(6):Art. 43, 35, 2013.

[LYS15] Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. Privacy-Preserving Genome-Wide Association Studies on Cloud Environment using Fully Homomorphic Encryption. 15:S1, 12 2015.

[MBG+16] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. NFLlib: NTT-Based Fast Lattice Library. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 341–356. Springer, 2016.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can Homomorphic Encryption be Practical? In Christian Cachin and Thomas Ristenpart, editors, *ACM Cloud Computing Security Workshop – CCSW*, pages 113–124. ACM, 2011.

[Pea00] Karl Pearson. X. On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is such that it can be Reasonably Supposed to have Arisen from Random Sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

[Rab81] Michael O Rabin. How to Exchange Secrets with Oblivious Transfer. Technical report, Aiken Computation Lab, Harvard University, 1981.

[SAM+17] Md Nazmus Sadat, Md Momin Al Aziz, Noman Mohammed, Feng Chen, Shuang Wang, and Xiaoqian Jiang. SAFETY: Secure GWAS in Federated Environment through a Hybrid Solution with Intel SGX and Homomorphic Encryption. *CoRR*, abs/1703.02577, 2017.

[SB16] Sean Simmons and Bonnie Berger. Realizing Privacy Preserving Genome-Wide Association Studies. *Bioinformatics*, 32(9):1293–1300, 2016.

[USF13] Caroline Uhlerop, Aleksandra Slavković, and Stephen E Fienberg. Privacy-Preserving Data Sharing for Genome-Wide Association Studies. *The Journal of privacy and confidentiality*, 5(1):137, 2013.

[ZBA15] Yihua Zhang, Marina Blanton, and Ghada Almashaqbeh. Secure Distributed Genome Analysis for GWAS and Sequence Comparison Computation. *BMC medical informatics and decision making*, 15(5):S4, 2015.