Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies

Oleksandr Tkachenko
TU Darmstadt, Germany
oleksandr.tkachenko@crisp-da.de

Christian Weinert
TU Darmstadt, Germany
christian.weinert@crisp-da.de

Thomas Schneider
TU Darmstadt, Germany
thomas.schneider@cs.tu-darmstadt.de

Kay Hamacher
TU Darmstadt, Germany
kay.hamacher@cysec.de

ABSTRACT

We present privacy-preserving solutions for *Genome-Wide Association Studies (GWAS)* based on *Secure Multi-Party Computation (SMPC)*. Using SMPC, we protect the privacy of patients when medical institutes collaborate for computing statistics on genomic data in a distributed fashion. Previous solutions for this task lack efficiency and/or use inadequate algorithms that are of limited practical value. Concretely, we optimize and implement multiple algorithms for the χ^2-, G-, and P-test in the ABY framework (Demmler et al., NDSS'15) and evaluate them in a distributed GWAS scenario.

Statistical tests generally require advanced mathematical operations. For operations that cannot be calculated in integer arithmetic, we make use of the existing IEEE 754 floating point arithmetic implementation in ABY (Demmler et al., CCS'15). To improve performance, we extend the mixed-protocol capabilities of ABY by optimizing and implementing the integer to floating point conversion protocols of Aliasgari et al. (NDSS'13), which may be of independent interest. Furthermore, we consider extended contingency tables for the χ^2- and G-test that use codeword counts instead of counts for only two alleles, thereby allowing for advanced, realistic analyses. Finally, we consider an outsourcing scenario where two non-colluding semi-trusted third parties process secret-shared input data from multiple institutes.

Our extensive evaluation shows, compared to the prior art of Constable et al. (BMC Medical Informatics and Decision Making'15), an improved run-time efficiency of the χ^2-test by up to factor 37x. We additionally demonstrate practicality in scenarios with millions of participants and hundreds of collaborating institutes.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; *Management and querying of encrypted data*; *Privacy protections*;

KEYWORDS

Privacy; secure computation; statistics; outsourcing; genetic algorithms; efficiency

ASIA CCS '18, June 4–8, 2018, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5576-6/18/06.
https://doi.org/10.1145/3196494.3196541

ACM Reference Format:
Oleksandr Tkachenko, Christian Weinert, Thomas Schneider, and Kay Hamacher. 2018. Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies. In *Proceedings of 2018 ACM Asia Conference on Computer and Communications Security (ASIA CCS '18).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3196494.3196541

1 INTRODUCTION

In 2000, the first human genome analysis took 9 months and cost 100 million USD. Nowadays, genome sequencing is much more affordable: it costs just about 2 000 USD and requires 15 minutes of computation [63]. Since genome sequencing is becoming more efficient and genomic data is getting collected widely, the research community pushes to conduct analyses on this data for many reasons. One possible application is investigating the associations between diseases and specific parts of the genome, which is increasingly used in epidemiology [35].

Genomic data is highly sensible and identifies its biological owner with a very high probability. There is also a kinship problem in the sense that the consent of an individual implies a (partial) decision for her relatives. Thus, genomic data must not be disclosed to the public. Disclosure could lead to disadvantaging people with predispositions to some diseases, also known as genetic discrimination [45]. For example, a health insurance could increase the fee or even decline the client's application because of "bad" genes.

Genomic data collectors usually sign contracts (so called "consent forms") with participants who provide their genomic information. These contracts allow the institutes to analyze the data, but not to share it with anyone else. However, since many diseases are very rare, the need of collaboration between institutes arises to perform distributed analyses on aggregated genomic data, e.g., in epidemiology. After aggregation from different sources, a sufficient amount of data for conducting statistical tests might be available [55]. Nonetheless, some contractual obligations currently forbid institutes to perform these distributed analyses as they are a clear violation of the data owner's privacy; or data centers require so-called "broad consent" — basically full authorization to do whatever a data holder wants — that ultimately reduces the number of voluntarily participating patients [57].

1.1 Motivation

In the past years, there were several attempts by the research community to provide cryptographic solutions for guaranteeing privacy

of data owners involved in distributed *Genome-Wide Association Studies (GWAS)*, several of them as part of the iDASH competition 2015[1]. These solutions allow institutes to securely share data on rare diseases and as a result to potentially improve the curative treatment of those diseases. Recently, it was recognized that the collection from some dozens of data centers (institutes) is necessary to derive new results, e.g., within the "Medizininformatik" initiative of the German Federal Ministry of Education and Research[2]. Thus, we long for a privacy-preserving solution that can even accommodate data from hundreds of data centers.

Some of the most promising solutions build on *Secure Multi-Party Computation (SMPC)* techniques. However, so far this approach appears to be rather slow and of limited use, since only data provided by relatively few participating patients can be processed. For example, in [14] the number of participants is restricted to 32 768, the precision is unsatisfying due to 16-bit floating point arithmetic, and it takes ∼22 min to compute the χ^2-test on ∼9 000 inputs.

Other approaches use k-anonymity [58], l-diversity [18, 56], or differential privacy [42, 51] to securely perform GWAS on distributed datasets. Nevertheless, several attacks were proposed shortly after: [25, 59, 61]. Also, these approaches do not seem to be suitable for GWAS because of their noise-based security, which reduces the utility of the results.

All current approaches for conducting privacy-preserving GWAS have in common that there is room for improvement in terms of performance. The ABY framework [17] appears to be well-suited for designing fast algorithms for the statistical tests applied in GWAS: it implements state-of-the-art and highly efficient protocols for *Secure Two-Party Computation (STPC)*. As the name indicates, ABY offers protocols in **A**rithmetic, **B**oolean, and **Y**ao sharing. It also supports switching between these sharing types during execution, which can be used to improve the overall efficiency of algorithms by choosing underlying protocols depending on where they perform best: Arithmetic sharing allows for free addition and cheap multiplication whereas the the other two differ in their performance for binary operations in different network settings.

Existing SMPC-based GWAS implementations also lack high precision and a possibility to take into account a large number of participants. These aspects can be improved by applying floating point arithmetic with increased bit-length, thereby delivering both, high precision and a wide range of possible values, e.g., 2^{-126} to 2^{127} for 32-bit floating point numbers. Fortunately, ABY already supports IEEE 754 floating point arithmetic with up to 64-bit precision [16].

So far, only a very limited number of statistics is available for privacy-preserving GWAS. This is related to the fact that more sophisticated statistics usually require more demanding operations which are not feasible in integer arithmetic. For example, the G-test requires the calculation of the logarithm.

Another issue with existing solutions for privacy-preserving GWAS is that they use counts for only two alleles to construct the contingency table for statistical tests. This is inadequate because of the information loss caused by the dimension reduction.

[1] http://www.humangenomeprivacy.org/2015/

[2] http://gesundheitsforschung-bmbf.de/de/medizininformatik-konzeptphase-3359.php

1.2 Our Contributions

In this work, we design a large set of algorithms for the χ^2-, G-, and P-test to be used in privacy-preserving distributed GWAS and present optimizations for their implementation in the ABY framework [17] as well as an extensive evaluation.

In prior art, Constable et al. [14] use only the χ^2-test and the *Minor Allele Frequency (MAF)* statistic, two columns in the contingency table, and perform only a very limited set of benchmarks in a two-party LAN setting. Their circuits operate only on 32-bit unsigned integer arithmetic and simulate 16-bit floating point arithmetic for the division operation required for the χ^2-test. In the following we detail how we overcome these restrictions.

Our contribution begins with the use of IEEE 754 floating point numbers and considering inter-protocol conversions in the implemented algorithms. In ABY, floating point operations can only be performed within Boolean and Yao sharing. However, we would like to use the more efficient Arithmetic sharing for addition and multiplication operations. To achieve this, we introduce optimized integer to floating point conversions based on the algorithms proposed by Aliasgari et al. [1]. By enriching ABY's mixed-protocol capabilities with this technique, we perform as many arithmetic operations within Arithmetic sharing as possible before switching to less efficient floating point computation.

Furthermore, we design three protocol variants for both, the χ^2- and the G-test, which are: 1) a straightforward algorithm implementation in Boolean sharing using 32-bit floating point numbers, 2) an optimization that performs addition and multiplication operations in Arithmetic sharing, and 3) an optimization that uses Arithmetic sharing as well, but not for multiplications (this allows for a much larger number of study participants, as we will see later).

Performing tests only on two allele counts, as done in [14], seems to be inappropriate for modern GWAS applications. Hence, we construct additional algorithms for the χ^2- and G-test considering k codeword counts instead of two allele counts.

The P-test is instantiated as a one-tailed threshold test using a pre-computed value from the χ^2-distribution with respect to some p-value. Concretely, we first compute the χ^2- or G-test and then verify whether this result is significant in terms of exceeding the given threshold.

We additionally consider an outsourcing scenario, where institutes securely outsource their genomic data to two non-colluding servers that run an STPC protocol. Here, each institute locally creates shares s_0 and s_1 of its aggregated genomic data and sends them securely to the corresponding non-colluding Semi-Trusted Third Parties (STTPs) T_0 and T_1. They collect the information from all participating institutes and calculate GWAS statistics on the aggregated data. After that, they send the resulting shares back to each institute which can locally reconstruct the result. Aggregating the data received from the institutes in Arithmetic sharing does not add any noticeable run-time costs to the STPC protocol.

Compared to [14], our implementation of the χ^2-test reduces the run-time by up to factor 37x. We additionally demonstrate practicality in scenarios with millions of participants and hundreds of collaborating institutes.

Table 1: Notation used throughout this paper.

T_0, T_1	Semi-trusted third parties performing STPC
$l[i]$	List operator referencing element i in list l
$l.e$	Access operator referring to element e in list l
$t \in \{A, B, Y\}$	Sharing type **A**rithmetic, **B**oolean, or **Y**ao
$\langle x \rangle_i^t$	Share of value x held by party i in sharing t
$\text{Rec}(\langle \text{x} \rangle_0^\text{t}, \langle \text{x} \rangle_1^\text{t})$	Reconstruction of value x from both shares in sharing t
$\text{Shr}_\text{i}^\text{t}(\text{x})$	Secret-sharing value x by party i in sharing t
$x \oplus y$ and $x \wedge y$	Bit-wise *XOR* and *AND* operation, respectively
$\langle z \rangle^t = \langle x \rangle^t \odot \langle y \rangle^t$	Operation on shares, $\odot$: $\langle x \rangle^t \times \langle y \rangle^t \mapsto \langle z \rangle^t$
$\langle x \rangle^s = t2s(\langle x \rangle^t)$	Conversion from sharing type t to s
$\langle 0 \rangle^t$, $\langle 1 \rangle^t$, $\langle 2 \rangle^t$	Secret-shared constant 0, 1, and 2, respectively
$\langle \mathbf{F}(\cdot) \rangle^t$	Secret-shared constant of local function output **F**

We summarize our main contributions as follows:

- Design and optimization of privacy-preserving algorithms for the χ^2-, G-, and P-test considering k codeword counts to be used in distributed GWAS.
- Implementation of different algorithm variants in ABY [17] using IEEE 754 floating point numbers and inter-protocol conversions including optimized integer to floating point conversions based on [1].
- Outsourcing computation to non-colluding STTPs, allowing hundreds of medical institutes to collaborate with negligible overhead compared to the two-party case.
- Extensive performance evaluation showing a run-time improvement of factor 37x over previous art [14] and demonstrating practicality in scenarios with millions of participants and hundreds of collaborating institutes.

2 PRELIMINARIES

In this section we introduce essential genome basics, the employed statistical tests, foundations of SMPC, and the ABY framework. Table 1 defines the notation used throughout this paper.

2.1 Genomic Primer

Genetics is the study of inheritance and genetic variability [50]. Genetic information is stored in a *Deoxyribonucleic Acid (DNA)* molecule. It consists of a phosphate backbone and *nucleotides*. A nucleotide consists of one of the four nucleobases: *Adenine (A), Cytosine (C), Thymine (T)*, or *Guanine (G)*. Nucleobases form *base pairs* according to the base pairing rules. The human genome consists of about 3 billion base pairs, however, genetic information of the human genome is more than 99% identical among individuals. Thus, it is reasonable to perform genetic analyses only on *Single-Nucleotide Polymorphisms (SNPs)*. SNPs are variations of base pairs in the DNA sequence. They are inheritable or inherited variants.

DNA builds structures called *chromosomes*, where each chromosome is a DNA molecule containing all or a part of the genome. The location of a gene in a chromosome is called *locus*. A genetic variant in a locus is called *allele*, which denotes the different forms of a gene located in the specific locus. Alleles determine alternative developing variants of a trait. They can result in observable differences in traits, i.e., *phenotypes* that imply observable effects.

Table 2: SNP contingency table. *a*, *b*, *c*, and *d* represent the number of observations of the respective allele in the respective group, i.e., the number of participants in the respective group for which the respective allele was observed.

	Allele 1	Allele 2	Total
Case Group	a	c	n_{G_1}
Control Group	b	d	n_{G_2}
Total	n_{A_1}	n_{A_2}	n

2.2 Genome-Wide Association Studies

Genome-Wide Association Studies (GWAS) are an approach of analyzing SNPs in order to find associations between diseases and genetic variants. For this purpose, the extent of linkage disequilibrium and the density of genetic markers should be adequate to make capturing the common variations in data possible.

The most commonly used strategy in GWAS is called *case-control group testing*. It is based on analyzing genetic data of case and control groups, where the case group denotes a group of people affected by a disease and the control group denotes a group of healthy people. GWAS are performed by applying statistical tests on the data of case and control groups in order to find discriminative patterns that allow to distinguish one group from the other. The representation of SNP data for GWAS in form of a contingency table is shown in Tab. 2.

2.3 Statistical Tests

Null Hypothesis Significance Testing (NHST) can be used for accepting or rejecting the *null hypothesis* (H_0), which captures the likelihood whether observed data can be explained by chance alone with respect to some *significance level* α.

Choosing $\alpha = 0.05$ is considered to be a good practice in statistics [21], but in GWAS much smaller values are used, e.g., $\alpha = 5{\cdot}10^{-8}$ [6], mainly due to so-called multiple-testing correction(s).

2.3.1 Chi-Squared Test. The term *χ^2-test* refers to a statistical hypothesis test group which assumes that the sample distribution is the χ^2 distribution if H_0 holds true. Its only parameter is the *degrees of freedom*, which is defined as $n = (\#columns - 1) \cdot (\#rows - 1)$ with respect to the contingency table. In this work, we apply the *χ^2-test of independence* and the *χ^2-goodness-of-fit* test.

Test of Independence. The test of independence is used for comparing multiple nominal variables and evaluating whether the proportions of these variables significantly differ. It is defined as

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

where $O_{i,j}$ are the observed values and $E_{i,j}$ are the expected values under H_0. While the observed value $O_{i,j}$ equals the value of cell (i, j) in the contingency table, the expected value $E_{i,j}$ is computed as $E_{i,j} = (n_{A_i} \cdot n_{G_j})/n$, where n_{A_i} is the number of observations in column i and n_{G_j} is the number of observations in row j.

Goodness-of-Fit Test. The goodness-of-fit test differs from the test of independence in the calculation of the expected value: In

GWAS, the expected value equals the number of observations in the corresponding cell in the control group. However, since case and control groups do not necessarily contain the same number of observations, we weight the observations by the number of observations in a group. The weighted observation is hence computed as $O'_{i,j} = O_{i,j} / \sum_{k=1}^{n} O_{i,k}$.

Computing the χ^2-value based on a p-value. The critical χ^2-test value is calculated from the inverted χ^2 *Cumulative Distribution Function (CDF)* given a p-value as described in Eq. 2 [43]:

$$x = F^{-1}(p, n) = \{x : F(x, n) = p\}$$
$$p = F(x, n) = \int_0^x \frac{t^{(n-1)/2} e^{-t/2}}{2^{n/2}\Gamma(n/2)} \mathrm{d}t, \tag{2}$$

where x is the critical χ^2-value, $\Gamma(\cdot)$ the Gamma function, p the p-value, and n the degrees of freedom parameter.

Essentially, the χ^2-test is an approximation of log-likelihood based statistics that was developed to avoid the calculation of logarithmic operations. Although the χ^2-test is easy to compute and mentioned in many textbooks as one of the most standard statistics, it slowly gets replaced by the G-test [44].

2.3.2 G-Test. The G-test is a likelihood-ratio or maximum likelihood statistical significance test. Compared to the χ^2-test it can better deal with a small number of observations (a crucial property when analyzing rare diseases with very few observations) and its result is additive, thus, the test can be performed stepwise. The calculation of the G-test is given in Eq. 3:

$$g = 2 \sum_{i,j} O_{i,j} \cdot \ln\left(\frac{O_{i,j}}{E_{i,j}}\right). \tag{3}$$

2.3.3 P-test. The P-test is a statistical test for proving that the result of a statistic exceeds a critical threshold that is based on significance level α. The threshold is defined as a parameter and corresponds to the probability of rejecting H_0 given that it is true. Hence, we reject H_0 if p-value $p < \alpha$. The calculation of the P-test is given in Eq. 4:

$$e_i = \begin{cases} 1 & \text{if } \mathbf{S}(X_i) > S_\alpha \\ 0 & \text{if } \mathbf{S}(X_i) \leq S_\alpha \end{cases}, \tag{4}$$

where e_i is an indicator flag for exceeding the critical threshold, $\mathbf{S}(X_i)$ is the result of statistic $\mathbf{S} \in \{\chi^2, g\}$ for SNP i, and S_α is the critical value for statistic $\mathbf{S}$. We compute S_α using a lookup table for p-values (computed as in Eq. 2) in the χ^2-distribution and reject H_0 if $e_i = 1$.

2.4 Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) allows multiple parties to jointly compute a function f on their inputs while keeping the respective inputs private and without using a trusted third party. For example, the parties could determine the highest salary among them without disclosing their own salary.

In this work we focus on *passive security*. This model yields very efficient protocols and protects against passive attacks, e.g., by insiders that have only read access. Moreover, constructing protocols assuming passive adversaries is most often an important step towards creating protocols secure against *active* adversaries that can arbitrarily deviate from the protocol in an attempt to cheat.

2.4.1 Oblivious Transfer. One of the most important building blocks for SMPC is a cryptographic protocol called *Oblivious Transfer (OT)*. The common 1-out-of-2 OT scheme was introduced by Rabin [49]: Here, the sender holds two messages m_0 and m_1, and the receiver holds a choice bit c. After the protocol, the receiver obtains m_c without disclosing the value of c and without gaining any information about m_{1-c}. *OT extension* protocols [3, 7, 28] allow to efficiently generate a large amount of OTs from a small amount of *base OTs* that require costly public-key cryptography.

2.4.2 Multiplication Triples. In order to evaluate interactive operations, e.g., in Arithmetic sharing (cf. §2.4.3) or the GMW protocol (cf. §2.4.4), one can use Multiplication Triples (MTs). They are defined as $\langle a \rangle^t \odot \langle b \rangle^t = \langle c \rangle^t$ with $\odot \in \{\cdot, \wedge\}$. As described in [17], MTs can be efficiently generated using OT extension [3].

2.4.3 Arithmetic Sharing. Arithmetic sharing denotes the additive sharing of an l-bit value x as the sum of two integers in the ring $\mathbb{Z}_{2^l}$ [4, 34]. More formally, $\langle x \rangle_0^A + \langle x \rangle_1^A \equiv x \mod 2^l$ with $\langle x \rangle_0^A, \langle x \rangle_1^A \in \mathbb{Z}_{2^l}$. For sharing value x, party P_i chooses a random value $r \in_R \mathbb{Z}_{2^l}$, computes $\langle x \rangle_i^A = x - r$ and sends r to P_{i-1}. P_{i-1} sets $\langle x \rangle_{i-1}^A = r$. The value can be reconstructed by sending $\langle x \rangle_{i-1}^A$ from P_{i-1} to P_i and P_i setting $x = \langle x \rangle_0^A + \langle x \rangle_1^A$.

In Arithmetic sharing it is possible to compute the addition and the multiplication of shared values. While addition can be computed locally, multiplication requires an interactive evaluation that can be performed using pre-computed multiplication triples.

2.4.4 GMW. In the *Goldreich-Micali-Wigderson (GMW)* [24] protocol, two parties P_0 and P_1 interactively and securely compute a function f that is represented as a boolean circuit. Each bit in the circuit corresponds to a so-called *wire* that is shared among the parties using a 2-out-of-2 secret sharing scheme. The value v of a wire is represented as $v = v_0 \oplus v_1$, where party P_i holds share v_i. Since XOR is an associative operation, XOR gates can be evaluated non-interactively. However, AND gates require an interactive evaluation that can be performed using pre-computed multiplication triples. Thus, not only the total number of AND gates in the circuit is a performance indicator, but also the AND-depth, which determines the number of required communication rounds.

2.4.5 Yao's Garbled Circuits (GC). In the protocol of Yao [64], like in the GMW protocol, two parties P_0 and P_1 securely compute a function f that is represented as a boolean circuit. However, in contrast to the GMW protocol, this protocol requires only a constant number of communication rounds. More precisely, P_0 acts as a *garbler* who garbles the circuit and its own inputs. P_1 on the other hand acts as the *evaluator* who evaluates the garbled circuit. P_1 receives its garbled inputs from P_0 using OTs. In Yao's GC protocol the evaluation of XOR gates is "free" [37], wheres the evaluation of AND gates requires transmitting and decrypting ciphertexts as part of the garbled circuit.

In [53], the authors find that in low latency / local networks GMW generally outperforms Yao, whereas in high latency / wide-area networks the constant round property might give Yao an advantage, depending on the function to be evaluated. In this work,

we focus on GMW since we use our outsourcing approach to shift secure computation to a low latency network setting.

2.5 The ABY Framework

ABY[3] [17] is a state-of-the-art and highly efficient framework for generic STPC. It allows users to construct circuits via C/C++ descriptions and supports their evaluation in three different protocols: **A**rithmetic (cf. §2.4.3), **B**oolean (aka GMW, cf. §2.4.4), and **Y**ao (cf. §2.4.5). Due to built-in conversion protocols, it is possible to switch the circuit evaluation protocol during execution. For example, some parts of a circuit are evaluated in Arithmetic and other parts are evaluated in Boolean sharing. By choosing the evaluation protocols for the operations they perform best, it is possible to reduce computation and communication costs significantly.

ABY supports IEEE 754 floating point arithmetic in Boolean and Yao sharing [16]. However, floating point arithmetic is very costly in STPC in terms of interactive operations, e.g., the addition of 32-bit and 64-bit floating point values requires ∼9x more AND gates than performing the operation on values of the respective bit-length in unsigned integer arithmetic.

The evaluation of a protocol in ABY is divided into two phases: *setup* and *online* phase. The setup phase pre-computes everything that is independent of the actual inputs, e.g., MTs, and the online phase performs interactive computations that depend on the inputs.

3 RELATED WORK

The research community started to move towards privacy-preserving solutions for distributed GWAS after it became clear that in unprotected studies the privacy of participants is threatened: Homer et al. [27] introduced an attack showing that the participation of an individual in GWAS can be revealed from the aggregated data. After that, many attacks were proposed to deanonymize individuals in distributed GWAS, e.g., [62, 71]. While the earlier attacks required extremely large databases for de-anonymization, finally a practical attack was constructed in [11].

Next, we describe existing approaches for mitigating such attacks grouped by the underlying techniques that guarantee privacy.

3.1 GWAS using Noise-based Approaches

Noise-based approaches are the most popular for ensuring privacy of individuals in distributed GWAS. Many solutions were proposed in recent years that apply *differential privacy*, e.g., [20, 29, 30, 51, 54, 60, 65, 66, 70]. Unfortunately, these methods have drawbacks. Adding noise to data generally reduces its utility and can influence the results. Therefore, researchers face difficulties in convincing regulatory authorities to accept such methods for clinical practice or drug development.

3.2 GWAS using Homomorphic Encryption

Homomorphic Encryption (HE) is a type of public-key cryptography that allows one to perform computations on ciphertexts and to obtain correct results after decryption.

Zhang et al. [68] proposed the FORESEE framework to fully outsource GWAS to the cloud. The main advantage of their framework is that it is able to compute the division operation unlike many other frameworks. They state their best run-times of about 52 ms for a single Single-Nucleotide Polymorphism (SNP) in the χ^2-test.

In [41], Lu et al. also perform GWAS in the cloud. They use a packing technique for the frequency table to improve the efficiency of the routine. They state 35 ms run-time for a χ^2-test evaluation on a single SNP with 10 000 observations.

Kim et al. [36] apply different fully homomorphic encryption schemes to compute the χ^2-test. Their best solution results in 5 ms amortized run-time for a single SNP. However, their evaluation is limited to 400 participants, does not consider network influences, and provides only 80-bit security.

Very recently, Bonte et al. [9] presented a solution based on the Fan-Vercauteren somewhat homomorphic encryption scheme [19]. However, compared to their second design based on additive sharing (cf. §3.3), it turns out that the HE solution is inferior: although the server performing the statistical computations on encrypted data requires only ∼1.5 ms almost independent of the number of participating institutes, the whole end-to-end computation takes more than a second to complete.

3.3 GWAS using Secure Computation

In [32], Kamm et al. propose that all institutes enter their whole collected genomic data into a storage system running a 3-out-of-3 additive sharing algorithm. On the shared data, they use Sharemind [8] to obtain results for statistics like the χ^2 test. For a single SNP with ∼1 000 observations the end-to-end computation requires 48 ms. They explicitly avoid floating point calculations, thereby limiting the selection of possible statistic calculations. Another noteworthy difference to our work is that we operate on *aggregated* data, thereby avoiding the outsourcing of the whole database.

Zhang et al. [67] propose a solution using Shamir's secret sharing scheme with a (n, t)-threshold configured s.t. at most one of the three parties can be corrupted. They propose a protocol for the privacy-preserving χ^2-test stating an amortized run-time of 4.5 ms for a SNP. However, they perform experiments with only 200 participants and the theoretic upper bound seems to be 2 048. Furthermore, they operate in a weaker security model (three parties with honest majority).

One of the most recent works using STPC techniques is by Constable et al. [14]. The statistical tests they use are the χ^2-test and the Minor Allele Frequency (MAF), which yields the most infrequent allele in a SNP. They simulate floating point arithmetic using 16-bit half-precision numbers. This is an important constraint since there are multiplications in the χ^2-test that can lead to an overflow or to precision interference. They pre-process SNPs locally by parsing the required number of SNPs and counting how many times alleles occurred in the observations. For example, the base pair AA will be parsed as two occurrences of the allele A, whereas AT and TA are both counted as one allele A and one T. However, by ignoring the positions of the alleles, they ignore most important genetic-structural information. Their implementation employs the PCF framework [38] for performing STPC and they run benchmarks on two servers in their local network. In our work, we improve the work of Constable et al. [14] and compare it to the performance

[3] https://github.com/encryptogroup/ABY

of our mixed-protocol algorithms (cf. §5.2). Furthermore, we extend the number of alleles in the contingency table and consider the G- and P-test as additional statistics. Finally, we consider an outsourcing scenario where multiple institutes perform SMPC on aggregated data by outsourcing the computation to two STTPs.

In independent and concurrent work, Bonte et al. [9] evaluate an implementation of the χ^2 significance test. More precisely, they consider a similar outsourcing approach as we do and employ the MASCOT framework [33] to perform the statistic computation in additive sharing between three parties. While their solution provides stronger security guarantees (active security with a dishonest majority), it has some shortcomings: The authors use transformation tricks to avoid floating point operations at all, thereby preventing an adaption of their approach to more advanced statistics like the G-test. Their benchmarks are performed on a single host without considering network influences. Furthermore, their evaluation includes only the online phase of the protocol and they do not report empirically on the heavyweight setup phase.

3.4 GWAS using Other Techniques

The authors of [12] propose to compute several GWAS statistics in a secure enclave using *Intel Software Guard Extensions (SGX)* [5]. Another recent solution [52] also employs SGX: here, data owners send their genotype counts Paillier-encrypted [15] to a central server which uses the homomorphic property to sum the inputs before performing statistical computations inside a secure enclave. The caveat with SGX-based solutions is that they require additional trust in the hardware manufacturer and the hardware itself.

4 DESIGN, OPTIMIZATION, AND IMPLEMENTATION

In this section we describe the design and optimization of the following algorithms applied to genetic data as well as the implementation of the required underlying techniques:

Pre-processing. The raw data (sequences of alleles, e.g., "SNP #1: AA TA …TT") must be processed and aggregated in a suitable format, e.g., the counts of alleles. For this purpose, we use a simple Python script not described further. We operate on data provided by the iDASH 2015[4] competition and replicate it in order to produce the required number of observations and SNPs.

Integer to floating point conversion gates. The idea behind these new conversion gates required for most of our algorithms is to perform the majority of all calculations on unsigned integers in Arithmetic sharing, which is very efficient. All remaining computations are performed after conversion to floating point numbers.

Statistical tests. We implement 4 versions of the χ^2- and G-test which differ in the way they are computed mathematically, the STPC protocols in which they are evaluated, and the number of alleles / codeword counts used in the contingency table. Additionally, we implement the P-test, which is generic and can be applied after each of the aforementioned tests in order to evaluate the statistical significance of the test result.

Outsourcing computation. Finally, we implement a scenario where multiple institutes send their secret-shared pre-processed genomic data to two Semi-Trusted Third Parties (STTPs), which aggregate the received data and perform the computation of the statistical test in the STPC protocol. After receiving the resulting shares, the institutes then reconstruct the result locally.

[4] http://www.humangenomeprivacy.org/2015/

$\langle [v], [p], [z], [s] \rangle \leftarrow$ **INT2FP**$([a], \gamma, l)$

1: $\lambda \leftarrow \gamma - 1$
2: $[s] \leftarrow$ **LTZ**$([a], \gamma)$
3: $[z] \leftarrow$ **EQZ**$([a], \gamma)$
4: $[a] \leftarrow (1 - 2[s])[a]$
5: $[a_{\gamma-1}], \ldots, [a_0] \leftarrow$ **BitDec**$([a], \lambda, \lambda)$
6: $[b_0], \ldots, [b_{\lambda-1}] \leftarrow$ **PreOR**$([a_{\lambda-1}], \ldots, [a_0])$
7: $[v] \leftarrow [a](1 + \sum_{i=0}^{\lambda-1} 2^i(1 - [b_i]))$
8: $[p] \leftarrow -(\lambda - \sum_{i=0}^{\lambda-1}[b_i])$
9: **if** $(\gamma - 1) > l$ **then** $[v] \leftarrow$ **Trunc**$([v], \gamma - 1, \gamma - l - 1)$
10: **else** $[v] \leftarrow 2^{l-\gamma+1}[v]$
11: $[p] \leftarrow ([p] + \gamma - 1 - l)(1 - [z])$
12: **return** $\langle [v], [p], [z], [s] \rangle$

Algorithm 1: Integer to floating point conversion algorithm from [1].

4.1 Integer to Floating Point Conversion

Multiple algorithms were proposed for computing floating point arithmetic in SMPC [1, 16, 26, 39, 47, 69] and in homomorphic encryption [22, 40]. They were implemented in the SMPC frameworks Sharemind [8], PICCO [69], and ABY [16].

For the conversion from integer to floating point we use the state-of-the-art algorithm introduced by Aliasgari et al. in [1] and adapt it to ABY. The algorithm is implemented in a simplified fashion and optimized for our purposes. We describe different optimization steps (labelled $O_0 \ldots O_3$) in the following.

The original version of the algorithm is shown in Alg. 1, where additive secret-sharing of value v is denoted as $[v]$ and vectors containing values v_1, v_2 as $\langle v_1, v_2 \rangle$. Input arguments are integer $[a]$, integer bit-length γ and floating-point representation bit-length l. The function returns a vector containing the floating point number $[v]$, exponent $[p]$, indicator zero $[z]$ of $[a]$ being 0 and indicator sign $[s]$ for $[a]$ being less than 0.

For our goals, we can immediately exclude the following parts of the protocol:

- Handling negative values (Lines 2 and 4), since ABY uses only unsigned integers.
- Bit decomposition (Line 5), as in ABY all values in boolean circuits are decomposed by default.
- Bit shifting/truncation (Lines 9 and 10), because we assume that integer numbers fit exactly into the representation. The reasoning behind this assumption is that the number of possible observations is naturally limited.

As an effect of these simplifications, we can also ignore the bit shifting in Line 11, because we fit exactly into the representation of the fraction. Moreover, the subtraction $(1-[z])$ in Line 11 can be simplified for boolean circuits by replacing it with the XOR operation, so we simplify it to $(1 \oplus [z])$ as $z \in \{0, 1\}$. Since the exponentiation in Line 7 is very costly, we replace it with *Multiplexer (MUX)* gates of pre-computed constant values of the form 2^i and use the inverted value b for selection. In addition to these simplifications, we provide

$\langle fp\_num \rangle^B \leftarrow$ **INT2FP**$(\langle int\_num \rangle^B, bitlen)$

1: $\langle is\_zero \rangle^B \leftarrow$ **EQZ**$(\langle int\_num \rangle^B)$
2: $\langle preor \rangle^B \leftarrow$ **PreOR**$(\langle int\_num \rangle^B)$
3: $\langle inv\_preor \rangle^B \leftarrow$ **Invert**$(\langle preor \rangle^B)$
4: $\langle frac \rangle^B \leftarrow$ **HammingWeight**$(\langle inv\_preor \rangle^B)$
5: $\langle frac \rangle^B \leftarrow$ **RShifter**$(\langle input \rangle^B, \langle frac \rangle^B)$
6: $\langle frac \rangle^B \leftarrow$ **Resize**$(\langle frac \rangle^B,$ **FractionSize**$(l))$
7: $\langle exponent \rangle^B \leftarrow$ **HammingWeight**$(\langle preor \rangle^B)$
8: $\langle exponent \rangle^B \leftarrow \langle exponent \rangle^B +$ **FPBias**$(bitlen))^B$
9: $\langle exponent \rangle^B \leftarrow$ **MUX**$(\langle exponent \rangle^B, \langle 0 \rangle^B, \langle is\_zero \rangle^B)$
10: $\langle fp\_num \rangle^B \leftarrow$ **Concatenate**$(\langle frac \rangle^B, \langle exponent \rangle^B, \langle 0 \rangle^B)$
11: **return** $\langle fp\_num \rangle^B$

Algorithm 2: Fully optimized integer to floating point conversion algorithm.

the following optimizations where optimization O_{i+1} includes the previous optimization O_i.

O_1: In Line 7, we still need λ additions. This implies communication overhead because of the large number of AND gates that are needed for addition in boolean circuits. To optimize this step, we can simply invert b and count its one-bits. For counting the one-bits (aka calculating the Hamming weight), we use the size-optimal circuit of [10]. This gives us the correct value for padding $[a]$ to the size of the significand. In ABY, an inversion gate can be evaluated without cryptographic computation.

O_2: In Line 11, there occurs a multiplication, which is costly in boolean circuits. Therefore, we compute $[p] \leftarrow [p] \cdot ([1] \oplus [z])$. This multiplication can be avoided by choosing either p or constant 0 using a MUX gate with selection bit z.

O_3: In Line 7, we need to multiply a by 2^k which, in boolean circuits, can be represented as a bit-shift of k-bits. Since there is no bit-shifter in the standard functionality of the ABY framework, we implement the barrel shifter introduced in [46]. By using a barrel shifter instead of multiplication, we significantly reduce the number of AND gates.

The final version of the algorithm is shown in Alg. 2. For the sake of simplicity, we will write $\langle fp\_num \rangle^B \leftarrow$ **INT2FP**$(\langle int\_num \rangle^A, bitlen)$ to indicate that the conversion **A2B**$(\langle int\_num \rangle^A)$ is performed prior to the actual integer to floating point conversion algorithm. The effects of the optimizations on the required number of gates are shown in Tab. 3. The last two columns denote an addition of 64-bit floating point numbers (FP_+) and unsigned integers ($UINT_+$) and are shown here for the purpose of comparison. Compared to the initial algorithm, the optimizations reduce the total number of gates and the number of AND gates by factor ~7x. Compared to the 64-bit floating point addition, a conversion operation requires a similar total number of gates, but ~6x fewer AND gates. This clearly indicates the benefit of using integer operations whenever possible before switching to floating point operations.

Table 3: Number of required gates for different integer to floating point conversions and addition operations.

	To 64-bit				To 32-bit	64-bit	
Gate	O_0	O_1	O_2	O_3	O_3	FP_+	$UINT_+$
AND	4 884	4 884	4 663	629	277	4 103	448
XOR	17 707	14 899	14 219	1 264	545	154	154
MUX	3 151	293	307	1 458	630	260	1
Total	25 745	20 076	19 189	3 351	1 452	4 517	768

4.2 Chi-Squared Test

We implement three different variants of the χ^2-test. With *standard* implementation we denote a straightforward boolean circuit operating continuously on 32-bit floating point numbers. We denote replacing all additions and multiplications with the corresponding integer operations in Arithmetic sharing and the use of integer to floating point conversions as *optimized* implementation. Replacing only additions is denoted as *large-scale* implementation.

The standard version implements Eq. 1. For the two optimized versions, we use the formula provided by [23]:

$$\chi^2 = \frac{(ad - bc)^2(a + b + c + d)}{(a + b)(c + d)(b + d)(a + c)}, \tag{5}$$

where a, b, c, and d are defined as in Tab. 2. This formula allows us to efficiently pre-compute almost all values in Arithmetic sharing before performing floating point operations. Since we operate on unsigned integers in ABY, we must prevent subtraction operations yielding negative values. For that purpose, we make use of the extended expression $(ad - bc)^2 = (ad)^2 + (bc)^2 - 2adbc$.

The optimized algorithm is given in Alg. 3 in App. A. We perform several additions and multiplications for determining the nominator and denominator. This leads to an integer overflow in Arithmetic sharing for even a small amount of observations. More precisely, for n observations of a SNP, we need to handle numbers of size $\left(n^4/16 - n^2\right) \cdot n$ in the worst case. In 32-bit arithmetic, this results in a limitation of 147 observations per SNP. Thus, we perform operations in 64-bit, thereby allowing up to 12 416 observations.

While this amount of observations is sufficient for the quantities of data available today, it is not what we consider a future-proof solution for large-scale analyses. Therefore, we propose the large-scale version of the algorithm in Alg. 5 in App. B. In this version, only additions are performed in Arithmetic sharing, while multiplications are performed in floating point arithmetic. Due to this change, we only need to make sure that the sum of all observations n (the right part of the nominator in Eq. 5) fits into the fraction when using our simplified floating point conversion protocol. As for the multiplications we can now rely on the built-in automatic scaling feature of the underlying IEEE 754 implementation.

As a result, we can handle up to $2^{23} - 1 = 8\,388\,607$ observations per SNP in 32-bit arithmetic and up to $2^{52} - 1$ observations in 64-bit arithmetic, limited by the bit-length of the IEEE 754 mantissa. The χ^2-test algorithm provided by the very recent work [9] must handle numbers up to size $n^6/16$. Thus, in their 128-bit additive secret sharing scheme they can only process up to 3 736 700 observations, assuming two's complement representation for negative numbers.

Table 4: Contingency table for the extended χ^2- and G-test.

	id_1	id_2	...	id_k	Total
Case Group	$obs_{1,1}$	$obs_{1,2}$	...	$obs_{1,k}$	n_{G_1}
Control Group	$obs_{2,1}$	$obs_{2,2}$	...	$obs_{2,k}$	n_{G_2}
Total	n_1	n_2	...	n_k	n

4.3 G-Test

Analogously to the χ^2-test, we implement three versions of the G-test: standard, optimized, and large-scale.

The standard algorithm is a straightforward implementation of Eq. 3. The only deviation is that we pre-compute the sums of observations for a SNP in the beginning in order to prevent repeated computation of the same values.

The optimized algorithm is given in Alg. 4 in App. A. As for the optimized χ^2-test, there are overflow problems. For the optimized G-test, we need to convert numbers of size n^2 to floating point representation in the worst case, where n is the total number of observations for a SNP. Thus, using 32-bit arithmetic, the floating point mantissa limits us to $\sqrt{2^{23}-1} \approx 2\,896$ observations.

While the usage of 64-bit arithmetic could increase the possible number of observations up to ~67 million, this turns out to be very inefficient due to the high cost of the 64-bit logarithm operation. Therefore, we present the large-scale version in Alg. 6 in App. B. In this version, we only pre-compute the sums in Arithmetic sharing and perform the remaining operations in Boolean sharing. As a result, like for the large-scale χ^2-test, we can handle up to $2^{23}-1 = 8\,388\,607$ observations per SNP in 32-bit arithmetic and potentially up to $2^{52}-1$ observations in 64-bit arithmetic.

4.4 Extended Chi-Squared and G-Test

The term "extended" test denotes a column extension of the original matrix (cf. Tab. 2) as shown in Tab. 4. We construct a contingency table with k codeword counts for both case and control groups. While the columns in Tab. 2 correspond to counts of single alleles in observed genotypes, here, columns correspond to codewords, i.e., genotypes or genotype sequences, whose length can vary depending on the task. The idea of this approach is to prevent the loss of information, e.g., the information that the first allele in a genotype is dominant and the second one is recessive will be considered. The loss occurs due to the dimension reduction of genotypes to only two allele counts.

The only difference in the statistic calculation is the calculation of observed and expected values. The calculation of the expected values for the χ^2 test of independence (cf. §2.3.1) causes a large overhead when using the extended contingency table. In order to reduce this overhead, the values are calculated according to the goodness-of-fit test (cf. §2.3.1). Hereafter, we evaluate the extended algorithms only in the large-scale configuration which achieves scalability in the number of possible participants.

4.5 P-Test

The calculation of the P-test builds on the previously discussed statistics. It checks whether the result is significant with respect to some significance level α (cf. §2.3.3). Since the threshold value is pre-calculated based on α, it is a public constant. To perform the P-test, we apply the Floating Point (FP) comparison operation ($\mathrm{FP}_{\mathrm{CMP}}$) that is already available in ABY to the threshold value and the outcome of the underlying statistic. The P-test yields 1 if the outcome exceeds the threshold and 0 otherwise. $\mathrm{FP}_{\mathrm{CMP}}$ requires 218 / 427 AND gates and 12 / 15 AND-depth for 32- / 64-bit values, respectively. Given these numbers, our comparison-based implementation is more efficient than adapting the masking technique proposed by [9] to the floating point case since this technique requires more expensive multiplications.

4.6 Circuit Complexities

We give the exact size and depth of the resulting circuits for the proposed algorithms with respect to the number of AND and multiplication gates for one SNP in Tab. 5. The reason for not considering other gate types is that they are "free" to compute in the respective sharings (cf. §2.4). Please note that the actual number of observations (i.e., participants) does not influence the circuit complexities. This is because we operate on aggregated values of a certain bit-length, thereby limiting only the maximum number of observations.

For the χ^2-test we see that the optimized version requires only ~27 % of the size of the standard version at similar depth. The large-scale version is ~20 % larger than the optimized version but requires only half the depth. The reduction in depth is because we require 64-bit arithmetic for the optimized version to allow for a reasonable number of observations, while for the large-scale version 32-bit arithmetic is sufficient.

For the G-test we see that the optimized version requires ~60 % of the size and depth of the standard version. Here, the difference in size is not as impressive as for the χ^2-test. This is due to the use of the complex logarithm operation which cannot be optimized and the fact that there is no optimized formula like for the χ^2-test (cf. Eq. 5). The large-scale version is ~30 % larger than the optimized version. However, for the G-test the depth for the large-scale and the optimized version is quite similar since we use 32-bit arithmetic for both of them.

For the extended versions of both algorithms we can observe that doubling the codeword counts results in doubling the size of the circuits. This reflects that we need to double the amount of observed and expected values. The depth however grows only by ~10 % when quadrupling the codeword counts. This is due to the fact that calculating observed and expected values can happen in parallel on the same layer of the circuit, while only the depth of the final addition tree increases.

All in all, the results in Tab. 5 confirm that the large-scale versions of the algorithms cost only marginally more than the optimized versions while bringing great improvements compared to the straightforward standard version.

4.7 Outsourcing Computation

As medical institutes are often not located next to each other, protocols with many rounds become impractical. Thus, the goal of outsourcing computation to two Semi-Trusted Third Parties (STTPs) is to move the protocol execution from the Wide Area Network (WAN) to the Local Area Network (LAN) setting. Furthermore, with this approach, $n > 2$ institutes can use more efficient STPC protocols.

Table 5: Size and depth of the circuits for the proposed algorithms with respect to the number of AND and multiplication gates for one SNP. With "Extended (x)" we denote the extended version with x codeword counts.

	Standard		Optimized		Large-scale		Extended (4)		Extended (16)		Extended (64)	
Test	Size	Depth	Size	Depth	Size	Depth	Size	Depth	Size	Depth	Size	Depth
χ^2	90 988	1 019	24 159	1 081	29 714	519	86 961	841	351 513	959	1 409 721	1 077
g	125 084	1 100	78 539	628	101 708	672	121 057	922	487 897	1 040	1 955 257	1 158

The STTPs do not receive any plaintexts, but are assumed to not collude. Institutes could, for example, agree on two different cloud service providers for operating the STTPs. Usually, the data centers of multiple such providers are located near an Internet Exchange Point (IXP), thereby enabling a low latency connection with high bandwidth, almost like in a local network. The competition among independent providers is a strong incentive to not collude. Depending on the specific application scenario, it would also be suitable to choose two the following entities for operating the STTPs: 1) (public) research agencies outside a consortium, 2) the ministry of health or other governmental agencies, or 3) consumer protection agencies. As long as their machines are co-located in different data centers near the same IXP, they can obtain the same performance as illustrated for the case of competing cloud service providers.

For outsourcing, we use the construction of [31] that turns any SMPC protocol into a provably secure outsourcing scheme which inherits the security guarantees from the underlying SMPC protocol. In our implementation, the institutes only send a single message to the two STTPs and each of these messages corresponds to a valid input of the institutes. Therefore, our protocols are secure against malicious institutes and one passively corrupted STTP (we assume that the two STTPs do not collude, see above).

We chose this model because it provides reasonable security while minimizing the associated costs, thereby encouraging real-world usage by institutes following budget-oriented guidelines: Security against passively corrupted non-colluding STTPs entails an acceptable performance overhead compared to the trivial solution of using a single trusted third party. Also, having only two STTPs, which in a real-world deployment would need to run different software stacks to reduce the attack surface and would be operated by different administrators, is cheaper than having three or more non-colluding servers with corresponding maintenance teams.

In detail, the outsourcing scheme consists of the following three steps (cf. Fig. 1 in App. C for a visualization):

Input sharing. We use existing internal ABY routines for locally creating shares to avoid the low-level programming required for handling shares of floating point numbers. After creating shares $\langle s \rangle_0^t$ and $\langle s \rangle_1^t$ for input value s locally, institute I sends these shares to the non-colluding STTPs T_0 and T_1 over respective secure channels (e.g., implemented via TLS).

Computing Statistics. The non-colluding servers receive shares from all institutes and compute statistic **S** interactively on share $\langle a \rangle^t$ which represents the aggregation of the received input data. The shares $\langle r \rangle_0^t$ and $\langle r \rangle_1^t$ of result r are output to the institutes.

Output reconstruction. After receiving the result shares via TLS, institute I computes $\text{Rec}(\langle r \rangle_0^t, \langle r \rangle_1^t)$ to reconstruct the plaintext result. For this task we again use existing internal ABY routines.

Hence, the statistic algorithms for outsourcing differ only in one aspect, namely in handling input and output gates. For this purpose we use the special shared input and output gates in ABY that were first introduced in [13]. These special gates require pre-shared values as input and output shared values, respectively.

5 PERFORMANCE EVALUATION

We run our benchmarks on two servers that are equipped with an Intel Core i7-4770K CPU @ 3.5 GHz and 16 GB of RAM. In the LAN setting the network bandwidth is 1 GB/s with ~0.1 ms latency. For the WAN setting we restrict the bandwidth to 100 MB/s and set the latency to 100 ms. Except when stated otherwise, our run-times and communication are the total costs for setup and online phase. The computational security parameter is set to 128 for all protocols.[5]

5.1 Benchmarking Results

In the following, we describe and analyze the benchmarking results of the implemented statistical tests in the LAN setting. This means that either two medical institutes or two non-colluding STTPs are connected in a low latency network with high bandwidth. A discussion on benchmarking results where two institutes are connected in the WAN setting can be found in App. E. The communication costs are analyzed in App. F.

We benchmark the algorithms with as many SNPs as possible while keeping the execution using only RAM to not influence run-times by using the swap space on the hard drive. Like for the circuit complexity analysis in §4.6, the evaluation results are completely independent of the number of observations (i.e., participants).

The run-times for all variants of the χ^2-, G-, and P-test are given in Tab. 6 and additionally visualized in Figs. 2 and 4 in App. D. These empirical results turn out to be in line with the analysis of the circuit complexities in §4.6 and scale linearly in the number of SNPs. Note that the given run-times for the statistical tests do not include input sharing and output reconstruction. This is because these procedures differ depending on whether the medical institutes interact directly or outsource computation. For the first case the overhead is separately given in Tab. 6 for the various bit-lengths and sharing types. The latter case is discussed in §5.1.5.

5.1.1 Chi-Squared Test. The standard version of the algorithm is significantly (2.2-3.5x) slower than the two optimized versions. This is not only due to the optimization using conversion gates, but also due to the more suitable equation for this task (cf. Eq. 5). The crucial

[5]To guarantee protection over longer periods of time, we refer to the standard keylength recommendations as summarized at https://www.keylength.com/. We note that compared to solutions based on HE, the overhead of our protocols grows only moderately when increasing the computational security parameter as our protocols are mostly based on symmetric cryptography.

Table 6: Run-times in seconds for the χ^2-, G-, and P-test algorithms in the LAN setting without input sharing and output reconstruction. Run-times for the extended algorithms are given for different codeword counts. Run-times for input sharing and output reconstruction in milliseconds are given for different bit-lengths and sharing types.

# SNPs	χ^2_s	χ^2_l	χ^2_o	χ^2_e					g_s	g_l	g_o	g_e					p		Input Sharing			Output Reconstruction		
				4	8	16	32	64				4	8	16	32	64	32-b B	64-b B	32-b A	64-b A	32-b B	1-b B	32-b B	64-b B
2^8	4.2	1.9	1.8	5	9	18	35	70	5.6	4.8	4.1	6	12	23	46	92	0.03	0.07	0.9	1.3	0.6	0.4	0.7	0.8
2^9	7.6	3.4	3.2	9	17	34	68	135	10	8.4	6.6	11	23	45	89	177	0.07	0.1	1.6	1.6	0.7	0.6	0.7	0.8
2^{10}	14	6.1	5.6	17	34	67	132	-	19	16	12	23	44	88	175	-	0.1	0.2	2.5	2.8	1.0	0.6	1.0	1.3
2^{11}	28	9.8	8.3	34	67	132	-	-	39	32	25	45	88	174	-	-	0.2	0.3	4.5	4.9	1.3	0.6	1.2	2.2
2^{12}	56	19	16	69	134	-	-	-	77	63	49	89	175	-	-	-	0.3	0.7	8.3	9.3	2.9	0.6	2.6	4.1
2^{13}	112	38	32	138	-	-	-	-	154	126	99	179	-	-	-	-	0.7	1.3	15.6	16.9	4.9	0.7	4.2	6.7
2^{14}	-	77	64	-	-	-	-	-	-	-	-	-	-	-	-	-	1.4	2.7	30.9	33.2	7.4	0.8	6.7	12.4
2^{15}	-	-	127	-	-	-	-	-	-	-	-	-	-	-	-	-	2.7	5.4	66.7	68.9	13.9	0.8	12.8	24.9

remark is that the large-scale algorithm is only slightly slower than the optimized algorithm, but allows a large number of participants in GWAS. More precisely, the optimized χ^2 algorithm is by factor 2.3-3.5x faster than the standard, and the large-scale algorithm is by factor 2.2-2.9x faster than the standard. The optimized algorithm is by only up to factor 1.2x faster than the large-scale. Summarizing, we can state a total amortized run-time of 3.88 ms per SNP for the optimized algorithm and 4.7 ms for the large-scale algorithm.

5.1.2 G-Test. The difference between the versions of the G-test is not as significant as for the χ^2-test since we use only one equation for all implementations. However, a tendency of the performance improvement can be seen directed to the optimized algorithms. The optimized algorithm is by factor 1.4-1.6x faster than the standard, and the large-scale algorithm is by factor 1.2x faster than the standard. The optimized algorithm is by factor 1.2-1.3x faster than the large-scale. Summarizing, we can state a total amortized run-time of 12 ms per SNP for the optimized algorithm and 15 ms for the large-scale algorithm.

5.1.3 Extended Chi-Squared and G-Test. The run-times for the extended versions scale not only linearly in the number of SNPs, but also in the number of codeword counts: doubling the number of SNPs results in roughly the same run-time as when doubling the number of codewords. The amortized total run-times are 17 ms / 264 ms for the extended χ^2-test and 22 ms / 346 ms for the extended G-test with 4 / 64 codewords, respectively.

5.1.4 P-Test. The run-time overheads for the P-test are given for both, 32- and 64-bit arithmetic (the 64-bit version is only used by the optimized χ^2-test). As well as the underlying statistics, the P-test has linear complexity in the number of SNPs. The amortized run-time of the P-test is 0.08 ms / 0.16 ms for 32- / 64-bit numbers, respectively, which is negligible compared to any of the underlying statistics. Consequently, one can efficiently apply the P-test to hide the exact results of the underlying statistic while still performing hypothesis testing.

5.1.5 Outsourcing Computation. In the outsourcing scenario described in §4.7, multiple institutes are connected to two non-colluding STTPs via WAN. The two STTPs are connected via LAN.

Input Sharing and Output Reconstruction. The time for creating shares locally using ABY routines takes much less than 1 % of the total run-time for all our algorithms and thus is completely negligible. More precisely, creating shares locally for 2^{15} SNPs in 32-bit Arithmetic / 64-bit Arithmetic / 32-bit Boolean sharing takes only 0.22 µs / 0.28 µs / 221 µs, respectively. The amortized run-time for institutes to locally reconstruct the results from the received shares is 5 µs / 8 µs per SNP for 32- / 64-bit shares, respectively, and 0.05 µs for revealing the 1-bit result of the P-test, which is also negligible compared to the total run-time of any evaluated protocol.

Secure Transfer of Shares. We analyzed the transfer times in the WAN setting using TLS[6] for securing the communication between institutes and STTPs s.t. an attacker cannot intercept both shares during transfer and reconstruct the secret data. We observed that the transfers have rather large variations regarding speed. For transferring data from 128 B up to 512 MB, the speed varies in the range 32-98 Mbit/s. Despite these variances, the transfer is still fast enough to have only a minor impact on the total run-time for any of the protocols. For example, also accounting for latency, it takes at most 350 ms to transfer 32-bit shares shares of a 2 by 2 contingency table (cf. Tab. 2) for a batch of 2^{15} SNPs and additional 163 ms to receive 32-bit result shares for all SNPs. However, when up to 256 institutes are engaged in the data transfer, it takes up to 32.1 s / 8.1 ms for sending / receiving shares, respectively. This example assumes that a TLS tunnel is already established, which is a one-time expense.

Computing Statistics. We ran our algorithms between the two non-colluding STTPs for different numbers of collaborating institutes to determine how much overhead an increased number of institutes entails. It is important to note that in this benchmark step the STTPs already received the input shares from all institutes. The number of SNPs is fixed to 256 and the maximum number of institutes is set to 256, such that the execution of all protocols still fits into RAM. We exclude the P-test from this benchmark because it has a constant overhead on top of the underlying statistic that depends only on the number of SNPs, which is constant here. The results are given in Tab. 7 and visualized in Fig. 3 in App. D.

The large-scale and optimized algorithms, where inputs are shared in Arithmetic sharing, have a nearly constant run-time on the number of involved institutes. This is due to non-interactive addition in Arithmetic sharing, i.e., no communication between the STTPs is required. The additionally required computation time however does not influence the results for even a 128-fold difference in the number of institutes. Thus, the run-times will be valid also for even a larger number of collaborators, because we can sum up all shares locally before executing the protocol in ABY. The run-time

[6] We use the stunnel proxy (https://www.stunnel.org/) for TLS support.

Table 7: Run-times in seconds for the χ^2- and G-test on 256 SNPs between two STTPs for different numbers of collaborating institutes.

# Institutes	χ^2_s	χ^2_l	χ^2_o	g_s	g_l	g_o
2	4.2	1.9	1.8	5.6	4.8	4.1
4	5.2	1.9	1.8	6.9	4.8	4.1
8	6.2	1.9	1.8	7.8	4.8	4.1
16	8.5	1.9	1.8	10	4.8	4.1
32	13	1.9	1.8	15	4.8	4.1
64	22	1.9	1.8	24	4.8	4.1
128	41	1.9	1.8	43	4.8	4.1
256	79	1.9	1.8	82	4.8	4.1

Table 8: Run-times in seconds for our χ^2-test algorithms compared to those of [14] in the LAN setting.

# SNPs	Constable et al. [14]	χ^2_s	χ^2_l	χ^2_o
311	47	4.9	2.2	2.1
9 330	1 342	128	44	36

of the standard algorithms on the other hand, which receive inputs in Boolean sharing and require interaction between the STTPs for the addition operations, grows linearly in the number of institutes.

5.2 Comparison with Constable et al. [14]

In Tab. 8, we compare the run-times of our χ^2 algorithms to the χ^2 algorithm introduced in [14], the only related work entirely relying on STPC techniques. Since their implementation is not publicly accessible, we need to rely on the benchmarks given in their paper. They only provide results for batches of 311 and 9 330 SNPs. Thus, we compare results for these SNP counts. Their benchmarking environment is quite similar to ours, except that their servers are equipped with slightly weaker Intel Core i5-750 CPUs @ 2.66 GHz.

The run-times of the algorithm introduced in [14] are always worse than ours, even compared to our slowest algorithm: the standard χ^2-test outperforms the solution of Constable et al. by factor 10x. Both, the optimized and the large-scale algorithm, have roughly 30x better run-times than [14]. Therefore, we assume their algorithms to be slower also in our benchmarking environment.

5.3 Comparison with Bonte et al. [9]

For the sake of completeness, we compare the run-times of our significance test based on the large-scale χ^2-test in Tab. 9 to those reported for the SMPC solution in the very recent work [9]. In [9] they report only the online phase of the protocol measured on a single machine with an Intel Core i5-3570K CPU @ 3.4 GHz, thereby omitting any network influences and not accounting for the offline phase. Thus, to make the results more comparable, we also run our protocols on a single machine and report only the online phase. It turns out that the online phase of our implementation is more than 10x faster than the online phase of [9]. However, please note that in this work we use a different computational model (two vs. three computation nodes) and a different security model (passive vs. active security). For a justification of our choice, we refer the reader to §4.7.

Table 9: Amortized run-times in milliseconds of the online phase of our significance test on χ^2-test basis compared with the results of [9] measured on localhost.

# Institutes	# Observations	[9]	Ours
20	200 000	2.2	0.19
40	400 000	2.3	0.19
60	600 000	2.3	0.19
80	800 000	2.5	0.19
100	1 000 000	2.4	0.19

6 CONCLUSION

Using and extending the secure two-party computation framework ABY [17] enabled us to construct and implement very efficient algorithms for the χ^2-, G-, and P-test that operate on IEEE 754 floating point numbers and outperform the best previous works in the area of privacy-preserving distributed GWAS by up to factor 37x. In addition, we implemented more realistic versions of the protocols that operate on codeword counts instead of counts of only two alleles, thereby preventing information loss. Finally, we considered an outsourcing scenario that allows hundreds of medical institutes to conduct research on securely outsourced data without noticeable overhead, thereby making our approach scalable.

As part of future work we envision implementing further statistical tests (e.g., Student's t-test [48] or Fisher's exact text [2]) and investigating floating-point-like number representations that might be better suited for secure statistical computations (e.g., [22]).

ACKNOWLEDGMENTS

We thank Sebastian Stammler for helpful discussions. We also thank the anonymous reviewers of ASIACCS'18 and our shepherd Manuel Barbosa for helpful comments on our paper. This work has been cofunded by the DFG as part of project E4 within the CRC 1119 CROSSING, and by the German Federal Ministry of Education and Research (BMBF) and by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP. Kay Hamacher gratefully acknowledges support and stimulating discussions within the HiGHmed project funded by the BMBF.

REFERENCES

[1] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. 2013. Secure computation on floating point numbers. In *NDSS*.
[2] F. Aminkeng, A. Bhavsar, H. Visscher, S. Rassekh, Y. Li, J. Lee, L. Brunham, H. Caron, E. van Dalen, and L. Kremer. 2015. A coding variant in RARG confers susceptibility to anthracycline-induced cardiotoxicity in childhood cancer. In *Nature Genetics*.
[3] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*.
[4] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. 2004. Private collaborative forecasting and benchmarking. In *WPES*.
[5] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A. Sadeghi, G. Scerri, and B. Warinschi. 2016. Secure multiparty computation from SGX. In *FC*.
[6] G. Barsh, G. Copenhaver, G. Gibson, and S. Williams. 2012. Guidelines for genome-wide association studies. In *PLoS Genet*.
[7] D. Beaver. 1996. Correlated pseudorandomness and the complexity of private computations. In *STOC*.
[8] D. Bogdanov, S. Laur, and J. Willemson. 2008. Sharemind: a framework for fast privacy-preserving computations. In *ESORICS*.
[9] C. Bonte, E. Makri, A. Ardeshirdavani, J. Simm, Y. Moreau, and F. Vercauteren. 2017. Privacy-preserving genome-wide association study is practical. In *Cryptology ePrint Archive: Report 2017/955*. http://ia.cr/2017/955.

[10] J. Boyar and R. Peralta. 2008. Tight bounds for the multiplicative complexity of symmetric functions. In *TCS*.

[11] R. Cai, Z. Hao, M. Winslett, X. Xiao, Y. Yang, Z. Zhang, and S. Zhou. 2015. Deterministic identification of specific individuals from GWAS results. In *Bioinformatics*.

[12] F. Chen, M. Dow, S. Ding, Y. Lu, X. Jiang, H. Tang, and S. Wang. 2016. PREMIX: privacy-preserving estimation of individual admixture. In *AMIA*.

[13] M. Chiesa, D. Demmler, M. Canini, M. Schapira, and T. Schneider. 2017. SIXPACK: Securing Internet eXchange Points Against Curious onlooKers. In *CoNEXT*.

[14] S. Constable, Y. Tang, S. Wang, X. Jiang, and S. Chapin. 2015. Privacy-preserving GWAS analysis on federated genomic datasets. In *BMC Medical Informatics and Decision Making*.

[15] I. Damgård, M. Jurik, and J. Nielsen. 2010. A generalization of Paillier's public-key system with applications to electronic voting. In *IJISS*.

[16] D. Demmler, G. Dessouky, F. Koushanfar, A. Sadeghi, T. Schneider, and S. Zeitouni. 2015. Automated synthesis of optimized circuits for secure computation. In *CCS*.

[17] D. Demmler, T. Schneider, and M. Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.

[18] C. Dwork, F. McSherry, K. Nissim, and A. Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *TCC*.

[19] J. Fan and F. Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. In *Cryptology ePrint Archive: Report 2012/144*. http://ia.cr/2012/144.

[20] S. Fienberg, A. Slavkovic, and C. Uhler. 2011. Privacy preserving GWAS data sharing. In *ICDMW*.

[21] R. Fisher. 1925. *Statistical methods for research workers*.

[22] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. 2010. Secure computations on non-integer values. In *WIFS*.

[23] D. Gifford. 2014. Foundations of Computational and Systems Biology. (2014). https://ocw.mit.edu/courses/biology/7-91j-foundations-of-computational-and-systems-biology-spring-2014/lecture-slides/MIT7_91JS14_Lecture20.pdf.

[24] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *STOC*.

[25] A. Gutmann, J. Wagner, Y. Ali, A. Allen, J. Arras, B. Atkinson, N. Farahany, A. Garza, C. Grady, and S. Hauser. 2012. Privacy and progress in whole genome sequencing. In *Presidential Committee for the Study of Bioethical Issues*.

[26] W. Henecka, A. Sadeghi, T. Schneider, and I. Wehrenberg. 2010. TASTY: tool for automating secure two-party computations. In *CCS*.

[27] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. Pearson, D. Stephan, S Nelson, and D. Craig. 2008. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. In *PLoS Genet*.

[28] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. 2003. Extending oblivious transfers efficiently. In *CRYPTO*.

[29] X. Jiang, Y. Zhao, X. Wang, B. Malin, S. Wang, L. Ohno-Machado, and H. Tang. 2014. A community assessment of privacy preserving techniques for human genomes. In *BMC Medical Informatics and Decision Making*.

[30] A. Johnson and V. Shmatikov. 2013. Privacy-preserving data exploration in genome-wide association studies. In *SIGKDD*.

[31] S. Kamara and M. Raykova. 2011. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds*.

[32] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo. 2013. A new way to protect privacy in large-scale genome-wide association studies. In *Bioinformatics*.

[33] M. Keller, E. Orsini, and P. Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *CCS*.

[34] F. Kerschbaum, T. Schneider, and A. Schröpfer. 2014. Automatic protocol selection in secure two-party computations. In *ACNS*.

[35] J. Kim, Y. Bai, and W. Pan. 2015. An Adaptive Association Test for Multiple Phenotypes with GWAS Summary Statistics. *Genetic Epidemiology*.

[36] M. Kim and K. Lauter. 2015. Private genome analysis through homomorphic encryption. In *BMC Medical Informatics and Decision Making*.

[37] V. Kolesnikov and T. Schneider. 2008. Improved garbled circuit: free XOR gates and applications. In *ICALP*.

[38] B. Kreuter, A. Shelat, B. Mood, and K. Butler. 2013. PCF: a Portable Circuit Format for scalable two-party secure computation. In *USENIX Security*.

[39] T. Krips and J. Willemson. 2014. Hybrid model of fixed and floating point numbers in secure multiparty computations. In *ISC*.

[40] X. Liu, R. Deng, W. Ding, R. Lu, and B. Qin. 2016. Privacy-preserving outsourced calculation on floating point numbers. In *TIFS*.

[41] W. Lu, Y. Yamada, and J. Sakuma. 2015. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. In *BMC Medical Informatics and Decision Making*.

[42] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. 2007. l-diversity: privacy beyond k-anonymity. In *TKDD*.

[43] MathWorks. 2017. Chi-square inverse cumulative distribution function. http://mathworks.com/help/stats/chi2inv.html.

[44] J. McDonald. 2009. *Handbook of biological statistics*.

[45] M. Naveed, E. Ayday, E. Clayton, J. Fellay, C. Gunter, J. Hubaux, B. Malin, and X. Wang. 2014. Privacy and security in the genomic era. In *CCS*.

[46] M. Pillmeier, M. Schulte, and E. Walters. 2002. Design alternatives for barrel shifters. In *SPIE*.

[47] P. Pullonen and S. Siim. 2015. Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In *FC*.

[48] E. Quillen, X. Chen, L. Almasy, F. Yang, H. He, X. Li, X. Wang, T. Liu, W. Hao, and H. Deng. 2014. ALDH2 is associated to alcohol dependence and is the major genetic determinant of "daily maximum drinks" in a GWAS study of an isolated rural Chinese sample. In *American Journal of Medical Genetics Part B: Neuropsychiatric Genetics*.

[49] M. Rabin. 1981. How to exchange secrets with oblivious transfer. In *Technical Report TR-81*.

[50] R. Rieger, A. Michaelis, and M. Green. 2012. *Glossary of genetics and cytogenetics: classical and molecular*.

[51] R. Rogers and D. Kifer. 2017. A new class of private Chi-square hypothesis tests. In *AISTATS*.

[52] M. Sadat, M. Al Aziz, N. Mohammed, F. Chen, S. Wang, and X. Jiang. 2017. SAFETY: Secure gwAs in Federated Environment Through a hYbrid solution with Intel SGX and Homomorphic Encryption. In *arXiv preprint 1703.02577*. https://arxiv.org/abs/1703.02577.

[53] T. Schneider and M. Zohner. 2013. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC*.

[54] S. Simmons and B. Berger. 2016. Realizing privacy preserving genome-wide association studies. In *Bioinformatics*.

[55] C. Spencer, Z. Su, P. Donnelly, and J. Marchini. 2009. Designing genome-wide association studies: sample size, power, imputation, and the choice of genotyping chip. In *PLoS Genet*.

[56] S. Stammler, S. Katzenbeisser, and K. Hamacher. 2016. Correcting Finite Sampling Issues in Entropy l-diversity. In *Privacy in Statistical Databases*.

[57] K. S. Steinsbekk, B. Kåre Myskja, and B. Solberg. 2013. Broad consent versus dynamic consent in biobank research: Is passive participation an ethical problem? *European Journal of Human Genetics*.

[58] L. Sweeney. 2002. k-anonymity: a model for protecting privacy. In *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*.

[59] L. Sweeney, A. Abu, and J. Winn. 2013. Identifying participants in the personal genome project by name. In *Data Privacy Lab, IQSS*.

[60] C. Uhler, A. Slavković, and S. Fienberg. 2013. Privacy-preserving data sharing for genome-wide association studies. In *JPC*.

[61] J. Vaidya, B. Shafiq, X. Jiang, and L. Ohno-Machado. 2013. Identifying inference attacks against healthcare data repositories. In *AMIA Summits on Translational Science*.

[62] R. Wang, Y. Li, X. Wang, H. Tang, and X. Zhou. 2009. Learning your identity and disease from research papers: information leaks in genome wide association study. In *CCS*.

[63] A. Weintraub. 2016. Inside Genomics Pioneer Craig Venter's Latest Production. In *Technology Review*.

[64] A. Yao. 1986. How to Generate and Exchange Secrets. In *FOCS*.

[65] F. Yu, S. Fienberg, A. Slavković, and C. Uhler. 2014. Scalable privacy-preserving data sharing methodology for genome-wide association studies. In *Journal of Biomedical Informatics*.

[66] F. Yu and Z. Ji. 2014. Scalable privacy-preserving data sharing methodology for genome-wide association studies: an application to iDASH healthcare privacy protection challenge. In *BMC Medical Informatics and Decision Making*.

[67] Y. Zhang, M. Blanton, and G. Almashaqbeh. 2015. Secure distributed genome analysis for GWAS and sequence comparison computation. In *BMC Medical Informatics and Decision Making*.

[68] Y. Zhang, W. Dai, X. Jiang, H. Xiong, and S. Wang. 2015. Foresee: fully outsourced secure genome study based on homomorphic encryption. In *BMC Medical Informatics and Decision Making*.

[69] Y. Zhang, A. Steele, and M. Blanton. 2013. PICCO: a general-purpose compiler for private distributed computation. In *CCS*.

[70] Y. Zhao, X. Wang, X. Jiang, L. Ohno-Machado, and H. Tang. 2014. Choosing blindly but wisely: differentially private solicitation of DNA datasets for disease marker discovery. In *Journal of the American Medical Informatics Association*.

[71] X Zhou, B. Peng, Y. Li, Y. Chen, H. Tang, and X. Wang. 2011. To release or not to release: evaluating information leaks in aggregate human-genome data. In *ESORICS*.

A OPTIMIZED ALGORITHMS

Our optimized algorithms for the χ^2- and G-test are given in Algs. 3 and 4, respectively. In Alg. 4, the function **PrecompSums** summarizes Lines 5 - 8 from Alg. 3. Based on the pre-computation result, **TotalObs** computes the total number of observations by adding sum_1 and sum_2. Furthermore, **SumAlleles** selects either sum_1 or sum_2, depending on the horizontal position of the current *entry* in the contingency table. Likewise, **SumGroups** selects either sum_3 or sum_4, depending on the vertical position.

result $\leftarrow \chi^2_o(snps)$

1: *result* $\leftarrow \emptyset$
2: **foreach** *snp* **in** *snps* **do**
3: $\langle ad\rangle^A \leftarrow snp.\langle a\rangle^A \cdot snp.\langle d\rangle^A$
4: $\langle bc\rangle^A \leftarrow snp.\langle b\rangle^A \cdot snp.\langle c\rangle^A$
5: $\langle sum_1\rangle^A \leftarrow snp.\langle a\rangle^A + snp.\langle b\rangle^A$
6: $\langle sum_2\rangle^A \leftarrow snp.\langle c\rangle^A + snp.\langle d\rangle^A$
7: $\langle sum_3\rangle^A \leftarrow snp.\langle b\rangle^A + snp.\langle d\rangle^A$
8: $\langle sum_4\rangle^A \leftarrow snp.\langle a\rangle^A + snp.\langle c\rangle^A$
9: $\langle nom_{left}\rangle^A \leftarrow (\langle ad\rangle^A)^2 + (\langle bc\rangle^A)^2 - \langle 2\rangle^A \cdot (\langle ad\rangle^A \cdot \langle bc\rangle^A)$
10: $\langle nom_{right}\rangle^A \leftarrow \langle sum_1\rangle^A + \langle sum_2\rangle^A$
11: $\langle nom\rangle^A \leftarrow \langle nom_{left}\rangle^A \cdot \langle nom_{right}\rangle^A$
12: $\langle denom\rangle^A \leftarrow (\langle sum_1\rangle^A \cdot \langle sum_2\rangle^A) \cdot (\langle sum_3\rangle^A \cdot \langle sum_4\rangle^A)$
13: $\langle nom\rangle^B \leftarrow$ **INT2FP**$(\langle nom\rangle^A, 64)$
14: $\langle denom\rangle^B \leftarrow$ **INT2FP**$(\langle denom\rangle^A, 64)$
15: **Append**$(result, \langle nom\rangle^B / \langle denom\rangle^B)$
16: **return** *result*

Algorithm 3: Optimized χ^2-test algorithm.

result $\leftarrow g_o(snps)$

1: *result* $\leftarrow \emptyset$
2: **foreach** *snp* **in** *snps* **do**
3: *snp_result* $\leftarrow \emptyset$
4: *sums* $\leftarrow$ **PrecompSums**(snp)
5: $\langle n\rangle^A \leftarrow$ **TotalObs**$(sums)$
6: $\langle n\rangle^B \leftarrow$ **INT2FP**$(\langle n\rangle^A, 32)$
7: **foreach** $\langle entry\rangle^A$ **in** *snp* **do**
8: $\langle allele\_obs\rangle^A \leftarrow$ **SumAlleles**$(\langle entry\rangle^A, sums)$
9: $\langle group\_obs\rangle^A \leftarrow$ **SumGroups**$(\langle entry\rangle^A, sums)$
10: $\langle exp\rangle^A \leftarrow \langle group\_obs\rangle^A \cdot \langle allele\_obs\rangle^A$
11: $\langle exp\rangle^B \leftarrow$ **INT2FP**$(\langle exp\rangle^A, 32) / \langle n\rangle^B$
12: $\langle obs\rangle^B \leftarrow$ **INT2FP**$(\langle entry\rangle^A, 32)$
13: $\langle entry\_res\rangle^B \leftarrow$ **LN**$(\langle obs\rangle^B / \langle exp\rangle^B)$
14: $\langle entry\_res\rangle^B \leftarrow \langle entry\_res\rangle^B \cdot \langle obs\rangle^B$
15: **Append**$(snp\_result, (\langle entry\_res\rangle^B))$
16: **Append**$(result, \sum_i \langle snp\_result[i]\rangle^B)$
17: **return** *result*

Algorithm 4: Optimized G-test algorithm.

B LARGE-SCALE ALGORITHMS

Our large-scale algorithms for the χ^2- and G-test are given in Algs. 5 and 6, respectively.

result $\leftarrow \chi^2_l(snps)$

1: *result* $\leftarrow \emptyset$
2: **foreach** *snp* **in** *snps* **do**
3: $\langle sum_1\rangle^A \leftarrow snp.\langle a\rangle^A + snp.\langle b\rangle^A$
4: $\langle sum_2\rangle^A \leftarrow snp.\langle c\rangle^A + snp.\langle d\rangle^A$
5: $\langle sum_3\rangle^A \leftarrow snp.\langle b\rangle^A + snp.\langle d\rangle^A$
6: $\langle sum_4\rangle^A \leftarrow snp.\langle a\rangle^A + snp.\langle c\rangle^A$
7: $\langle n\rangle^A \leftarrow \langle sum_1\rangle^A + \langle sum_2\rangle^A$
8: $\langle a\rangle^B \leftarrow$ **INT2FP**$(snp.\langle a\rangle^A, 32)$
9: $\langle b\rangle^B \leftarrow$ **INT2FP**$(snp.\langle b\rangle^A, 32)$
10: $\langle c\rangle^B \leftarrow$ **INT2FP**$(snp.\langle c\rangle^A, 32)$
11: $\langle d\rangle^B \leftarrow$ **INT2FP**$(snp.\langle d\rangle^A, 32)$
12: **for** i **in** $1:4$ **do**
13: $\langle sum_i\rangle^B \leftarrow$ **INT2FP**$(\langle sum_i\rangle^A, 32)$
14: $\langle n\rangle^B \leftarrow$ **INT2FP**$(\langle n\rangle^A, 32)$
15: $\langle ad\rangle^B \leftarrow \langle a\rangle^B \cdot \langle d\rangle^B$
16: $\langle bc\rangle^B \leftarrow \langle b\rangle^B \cdot \langle c\rangle^B$
17: $\langle nom_{left}\rangle^B \leftarrow (\langle ad\rangle^B - \langle bc\rangle^B)^2$
18: $\langle nom\rangle^B \leftarrow \langle nom_{left}\rangle^B \cdot \langle n\rangle^B$
19: $\langle denom\rangle^B \leftarrow (\langle sum_1\rangle^B \cdot \langle sum_2\rangle^B) \cdot (\langle sum_3\rangle^B \cdot \langle sum_4\rangle^B)$
20: **Append**$(result, \langle nom\rangle^B / \langle denom\rangle^B)$
21: **return** *result*

Algorithm 5: Large-scale χ^2-test algorithm.

result $\leftarrow g_l(snps)$

1: *result* $\leftarrow \emptyset$
2: **foreach** *snp* **in** *snps* **do**
3: *snp_result* $\leftarrow \emptyset$
4: *sums* $\leftarrow$ **PrecompSums**(snp)
5: $\langle n\rangle^A \leftarrow$ **TotalObs**$(sums)$
6: $\langle n\rangle^B \leftarrow$ **INT2FP**$(\langle n\rangle^A, 32)$
7: **for** i **in** $1:4$ **do**
8: $\langle sums[i]\rangle^B \leftarrow$ **INT2FP**$(\langle sums[i]\rangle^A, 32)$
9: **foreach** $\langle entry\rangle^A$ **in** *snp* **do**
10: $\langle entry\rangle^B \leftarrow$ **INT2FP**$(\langle entry\rangle^A, 32)$
11: $\langle allele\_obs\rangle^B \leftarrow$ **SumAlleles**$(\langle entry\rangle^B, sums)$
12: $\langle group\_obs\rangle^B \leftarrow$ **SumGroups**$(\langle entry\rangle^B, sums)$
13: $\langle exp\rangle^B \leftarrow \langle group\_obs\rangle^B \cdot \langle allele\_obs\rangle^B / \langle n\rangle^B$
14: $\langle obs\rangle^B \leftarrow \langle entry\rangle^B$
15: $\langle entry\_res\rangle^B \leftarrow \ln(\langle obs\rangle^B / \langle exp\rangle^B)$
16: $\langle entry\_res\rangle^B \leftarrow \langle entry\_res\rangle^B \cdot \langle obs\rangle^B$
17: **Append**$(snp\_result, (\langle entry\_res\rangle^B))$
18: **Append**$(result, \sum_i \langle snp\_result[i]\rangle^B)$
19: **return** *result*

Algorithm 6: Large-scale G-test algorithm.

C OUTSOURCING SCHEME

The three steps of the outsourcing scheme discussed in §4.7 are visualized in Fig. 1.

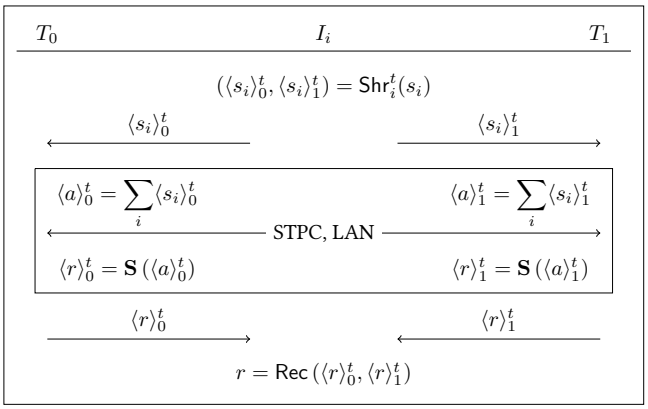

Figure 1: Outsourcing scheme for computing statistic S on aggregate data received from multiple parties.

D LAN BENCHMARKING

In Figs. 2 and 4 we depict the run-times from Tab. 6 for the different versions of the χ^2- and G-test algorithms in the LAN setting, respectively. Note again that these run-times do not include input sharing and output reconstruction.

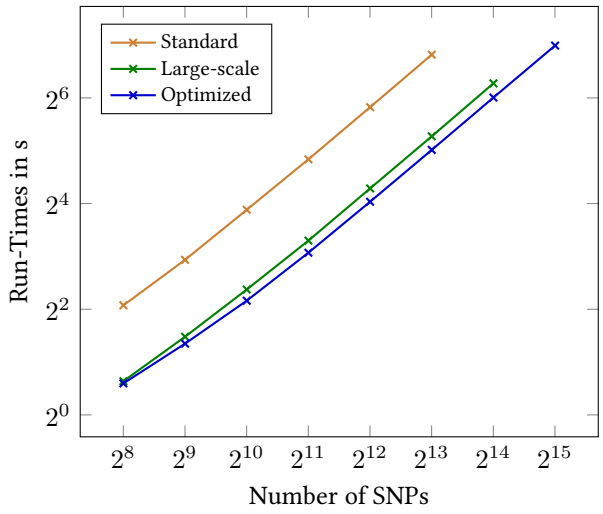

Figure 2: Run-times of the χ^2-test algorithms.

In Fig. 3 we depict the run-times from Tab. 7 for the the different versions of the χ^2- and G-test algorithms computed between two non-colluding STTPs on 256 SNPs. Note again that these run-times do not include secure transfer of the input and output shares.

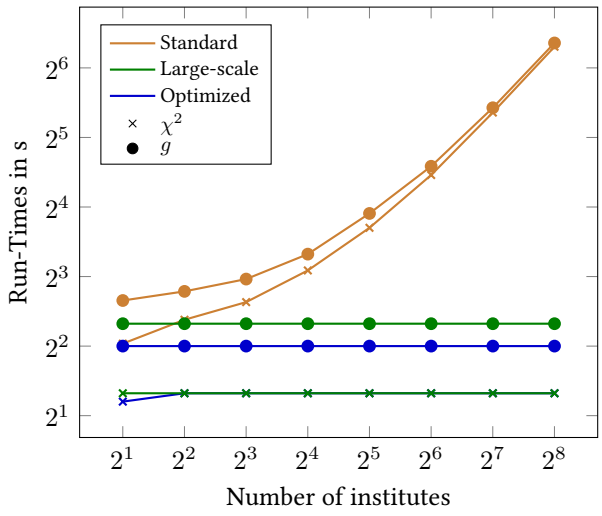

Figure 3: Run-times of the χ^2- and G-test on 256 SNPs between two STTPs for different numbers of collaborating institutes.

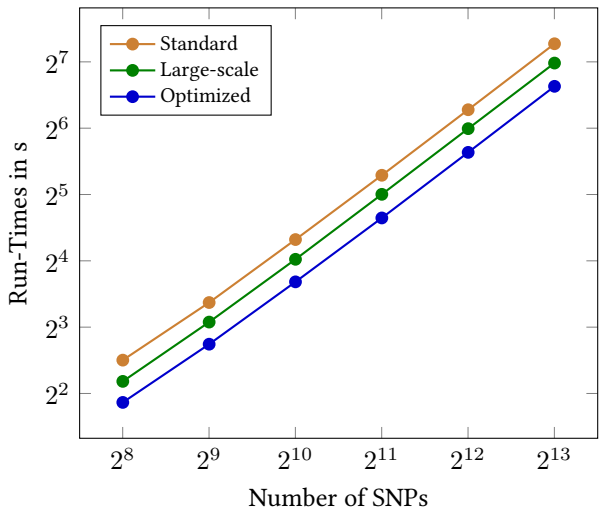

Figure 4: Run-times of the G-test algorithms.

E WAN BENCHMARKING

In Fig. 5, we depict the benchmarking results of our χ^2-test algorithms in the WAN setting and compare them to the LAN results.

We evaluated the algorithms both with the GMW and Yao's GC protocol (cf. §2.4.4 and §2.4.5, respectively). To do so, we replaced Boolean sharing with Yao sharing in all our algorithms. However, we found that GMW outperforms Yao's GC protocol despite a non-constant number of communication rounds. Thus, we report only results using Boolean sharing.

In our benchmarks, the algorithms were approximately by an order of magnitude slower in the WAN setting than in the LAN setting. This substantial difference was motivation to find a way for shifting the use of our algorithms to the LAN setting for spatially distant institutes, resulting in the outsourcing scenario discussed in §4.7 and §5.1.5. Interestingly, in the WAN setting the large-scale χ^2-test algorithm performs better than the optimized version due to the by factor ~2x lower AND-depth (cf. §4.6).

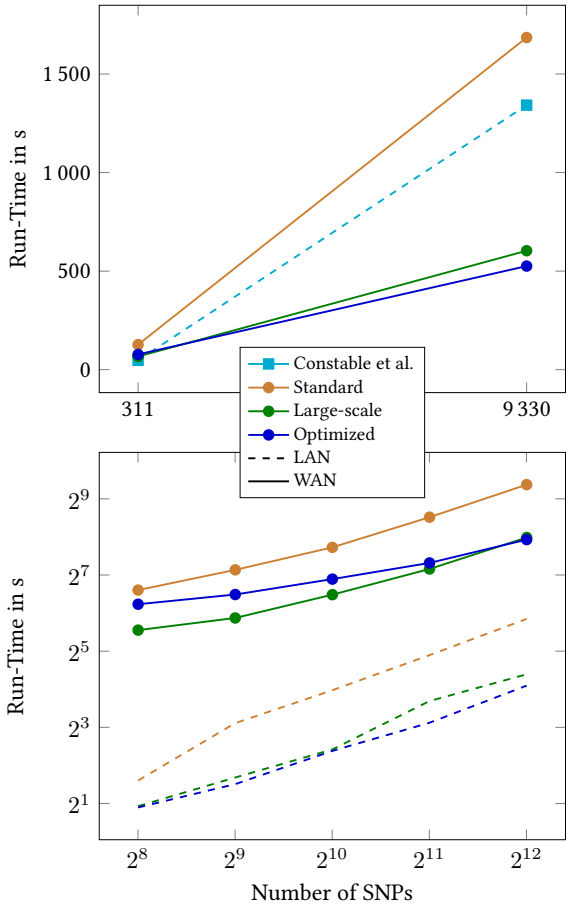

Figure 5: Run-times of the χ^2-test algorithms in the LAN and WAN setting. The upper graph shows that our optimized and large-scale algorithms are even faster in the WAN setting than the algorithm by Constable et al. [14] in the LAN setting.

F COMMUNICATION COSTS

The total amortized communication costs for performing a test on one SNP between two institutes or between two STTPs are shown in Tab. 10. Communication is almost equally distributed among the two parties. Hence, each party sends and receives about half the amount of data given in the table.

Table 10: Total amortized communication in megabytes of our χ^2-, G-, and P-test algorithms for one SNP.

χ^2_s	χ^2_l	χ^2_o	g_s	g_l	g_o	p (32-bit)	p (64-bit)
2.95	0.98	0.82	4.06	3.32	2.63	0.01	0.02

In Fig. 6 we depict the communication costs for the different versions of the χ^2- and G-test algorithms. Note that the communication costs are independent of the network setting and whether the protocols are run directly between two institutes or in the outsourcing scenario.

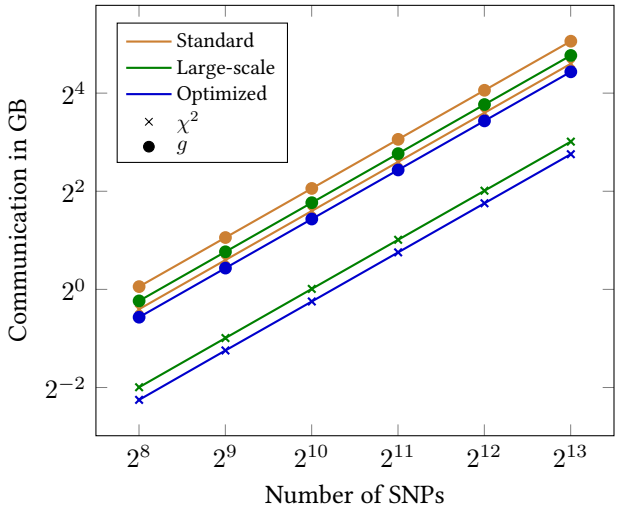

Figure 6: Communication costs of our χ^2- and G-test algorithms for different numbers of SNPs.

In Tab. 11 we give the amortized communication costs of the extended χ^2- and G-test for different codeword counts. Obviously, the algorithms scale linearly in the number of codeword counts.

Table 11: Total amortized communication in megabytes of our extended χ^2- and G-test algorithms for different codeword counts and one SNP.

# Codeword Counts	4	8	16	32	64	128	256
χ^2_e	2.8	5.7	11	23	46	92	184
g_e	3.9	7.9	15	31	63	127	255