Private Computation on Encrypted Genomic Data

Kristin Lauter, Microsoft Research
Adriana López-Alt[*], New York University
Michael Naehrig, Microsoft Research

Abstract

A number of databases around the world currently host a wealth of genomic data that is invaluable to researchers conducting a variety of genomic studies. However, patients who volunteer their genomic data run the risk of privacy invasion. In this work, we give a cryptographic solution to this problem: to maintain patient privacy, we propose encrypting all genomic data in the database. To allow meaningful computation on the encrypted data, we propose using a homomorphic encryption scheme.

Specifically, we take basic genomic algorithms which are commonly used in genetic association studies and show how they can be made to work on encrypted genotype and phenotype data. In particular, we consider the Pearson Goodness-of-Fit test, the D' and r^2-measures of linkage disequilibrium, the Estimation Maximization (EM) algorithm for haplotyping, and the Cochran-Armitage Test for Trend. We also provide performance numbers for running these algorithms on encrypted data.

1 Introduction

As the cost of sequencing the human genome drops, more and more genomic data will become available for scientific study. At the same time, researchers are developing new methods for analyzing genomic data across populations to look for patterns and find correlations. Such research may help identify genetic risk factors for disease, suggest treatments, or find cures. But to make this data available for scientific study, patients expose themselves to risks from invasion of privacy [ADCHT13]. Even when the data is anonymized, individual patients' genomic data can be re-identified [GMG$^+$13, WLW$^+$09] and can furthermore expose close relatives to similar risks [HAHT13].

A number of databases to host genomic data for research have been created and currently house a wealth of genomic data, for example the 1,000 Genomes Project [TGP], the International Cancer Genome Consortium (ICGC) [ICG], and the International Rare Diseases Research Consortium (IRDiRC) [IRD]. There are also a number of shared research databases which house de-identified genomic sequence data such as the eMERGE Network [MCC$^+$11], the Database of Genotypes and Phenotypes [dbG], the European Bioinformatics Institute [EBI], and the DNA Databank of Japan [Jap].

Various approaches to protecting genomic privacy while allowing research on the data include policy-based solutions, de-identification of data, approximate query-answering, and technological solutions based on cryptography. Emerging cryptographic solutions are quickly becoming more relevant. Encryption is a tool which essentially allows one to seal data in a metaphorical vault,

[*]Research conducted while visiting Microsoft Research.

which can only be opened by somebody holding the secret decryption key. *Homomorphic* Encryption (HE) allows other parties to operate on the data without possession of the secret key (metaphorically sticking their hands into the vault via a glove box and manipulating the data). *Fully* Homomorphic Encryption (FHE) allows the evaluation of *any* function on encrypted data but current implementations are widely inefficient. More practical variants of HE schemes allow for only a fixed amount of computation on encrypted data while still ensuring correctness and security. In particular, practical HE schemes allow for evaluation of polynomials of small degree.

In this work, we take basic genomic algorithms which are commonly used in genome wide association studies (GWAS) and show how they can be made to work on encrypted data using HE. We find a number of statistical algorithms which can be evaluated with polynomials of small degree, including the Pearson Goodness-of-Fit or Chi-Squared Test to test for deviation from Hardy-Weinberg equilibrium, the D' and r^2 measures of linkage disequilibrium to test for association in the genotypes at two loci in a genome, the Estimation Maximization (EM) Algorithm to estimate haplotype frequencies from genotype counts, and the Cochran-Armitage Test for Trend (CATT) to determine if a candidate allele is associated to a disease.

In our approach, these statistics are computed from encrypted genotype and phenotype counts in a population. Thus for a database containing encrypted phenotypes and genotypes, we consider two stages: in the first stage encrypted phenotype and genotype counts are computed using only simple additions. The parameters of the encryption scheme, as well as the running time of the computation in this stage depend on the size of the population sample being considered.[1] The second stage takes as input the encrypted genotype and phenotype counts obtained in the first stage and computes the output of the statistical algorithms mentioned above. In this stage the runtime of the statistical algorithms *does not depend* on the size of the population sample and only depends on the parameter set needed for the computation. Table 2 gives the timings to evaluate the statistical algorithms on encrypted genotype and phenotype counts. For example, the Cochran Armitage Test for Trend takes 0.94 seconds at the smaller parameter size and 3.63 seconds at the larger parameter size.

Genomic Databases: Hosted by a Trusted Party, Stored in an Untrusted Cloud. It is important to note that in this work we are considering single-key homomorphic encryption, which means that all data is encrypted under the same symmetric or asymmetric encryption key. To see how this can be used to protect privacy in genome databases as described above, consider the following scenario which captures one of the challenges facing government and research organizations currently deploying large-scale genomic databases for research.

A global alliance of government agencies, research institutes, and hospitals wants to pool all their patients' genomic data to make available for research. A common infrastructure is required to host all these data sets, and to handle the demands of distributed storage, providing a low cost solution which is scalable, elastic, efficient, and secure. These are the arguments for using commercial cloud computing infrastructure made by the Global Alliance [Glo13, p.17] in their proposal. Thus we arrive at the following requirement: data collected by a trusted host or hosts, such as a hospital or research facility, may need to be stored and processed in an untrusted cloud, to enable access and sharing across multiple types of boundaries. The mutually trusting data owners, i.e. the hospital or hospitals, can encrypt all data under a single key using homomorphic encryption. The cloud can then process queries on the encrypted data from arbitrary entities such as member organizations,

[1]The running time is linear in the population size for a fixed parameter set. For larger population sizes, parameters need to be increased and performance degrades, but not by a large factor (see Table 1 for a comparison of the running times for two typical parameter sets).

registered individual researchers, clinicians etc. The cloud can return encrypted results in response to queries, and the trusted party can provide decryptions to registered parties according to some policy governing allowable queries. Note that the policy should not allow arbitrary queries, since this would expose the data to the same re-identification risks that an unencrypted public database faces. However, with a reasonable policy, this would allow researchers to study large data sets from multiple sources without making them publicly available to the researchers who query them.

Related Work. Much of the related work on genomic privacy focuses on the problem of pattern-matching for genomic sequences, which is quite different from the statistical algorithms we analyze here. Actually the circuits for pattern matching and edit distance are much deeper than those considered here, so less suitable as an efficient application of HE. On the other hand, De Cristofaro et al. [DCFT13] present an algorithm for private substring-matching which is extremely efficient. In another approach, Blanton et al. [BAFM12] efficiently carry out sequence comparisons using garbled circuits. Finally, Ayday et al. [ARH13] show how to use *additively* homomorphic encryption to predict disease susceptibility while preserving patient privacy.

Differential privacy techniques have also been investigated in several recent papers [FSU11, JS13]. Fienberg et al. [FSU11] propose releasing differentially private minor allele frequencies, chi-square statistics and p-values as well as a differentially-private approach to penalized logistic regression (see e.g. [PH08]). Johnson and Shmatikov [JS13] present a set of privacy-preserving data mining algorithms for GWAS datasets based on differential privacy techniques.

Finally a recent line of work investigating practical applications of HE to outsourcing computation on encrypted data has led to the present paper. Lauter et al. [LNV11] introduce the idea of medical applications of HE, with optimizations, concrete parameters, and performance numbers. Graepel et al. [GLN13] apply HE to machine learning, both to train a model on encrypted data, and to give encrypted predictions based on an encrypted learned model. Bos et al. [BLN14] give optimized performance numbers for HE and a particular application in health care to predictive analysis, along with an algorithm for automatically selecting parameters. Yasuda et al. [YSK$^+$13] give an application of HE to secure pattern matching.

2 Statistical Algorithms in Genetic Association Studies

In this section, we detail common statistical algorithms used in genetic association studies. We consider the Pearson Goodness-Of-Fit test, which is used to determine deviation from Hardy-Weinberg Equilibrium (HWE) (Section 2.1), the D' and r^2 measures of linkage disequilibrium, as well as the Estimation-Maximization (EM) algorithm for haplotyping (Section 2.2), and the Cochran-Armitage Test for Trend (CATT) used in case-control studies (Section 2.3).

2.1 Hardy-Weinberg Equilibrium and the Pearson Goodness-Of-Fit Test

We begin by describing the Pearson Goodness-Of-Fit test, a test frequently used to determine whether a gene is in Hardy-Weinberg Equilibrium (HWE). We first review the notion of HWE and then describe the Pearson test.

Hardy-Weinberg Equilibrium (HWE). A gene is said to be in HWE if its allele frequencies are independent. More specifically, suppose A and a are two alleles of the gene being considered, and let $N_{AA},\ N_{Aa},\ N_{aa}$ denote the observed population counts for genotypes $AA,\ Aa,\ aa$, respectively.

Also let N be the total number of people in the sample population; that is, $N \overset{\mathsf{def}}{=} N_{AA} + N_{Aa} + N_{aa}$. With this notation, the corresponding frequencies of the genotypes AA, Aa, aa are given by

$$p_{AA} \overset{\mathsf{def}}{=} \frac{N_{AA}}{N} \ , \quad p_{Aa} \overset{\mathsf{def}}{=} \frac{N_{Aa}}{N} \ , \quad p_{aa} \overset{\mathsf{def}}{=} \frac{N_{aa}}{N} \ .$$

Moreover, the frequencies of the alleles A and a are given by

$$p_A \overset{\mathsf{def}}{=} \frac{2N_{AA} + N_{Aa}}{2N} \ , \quad p_a \overset{\mathsf{def}}{=} \frac{2N_{aa} + N_{Aa}}{2N} = 1 - p_A \ ,$$

since each count of genotype AA contributes two A alleles, each count of genotype aa contributes two a alleles, each count of genotype Aa contributes one A allele and one a allele, and the total number of alleles in a sample of N people is $2N$.

The gene is said to be in equilibrium if these frequencies are independent, or in other words, if

$$p_{AA} = p_A^2 \ , \quad p_{Aa} = 2p_A p_a \ , \quad p_{aa} = p_a^2 \ .$$

When a gene is in equilibrium, its allele frequencies stay the same from generation to generation unless perturbed by evolutionary influences. Researchers test for HWE as a way to test for data quality, and might discard loci that deviate significantly from equilibrium.

Pearson Goodness-Of-Fit Test. The main observation made by the Pearson Goodness-of-Fit test is that if the alleles are independent (i.e. if the gene is in equilibrium) then we expect the observed counts to be

$$E_{AA} \overset{\mathsf{def}}{=} N p_A^2 \ , \quad E_{Aa} \overset{\mathsf{def}}{=} 2N p_A p_a \ , \quad E_{aa} \overset{\mathsf{def}}{=} N p_a^2 \ .$$

Thus, deviation from equilibrium can be determined by comparing the X^2 test-statistic below to the χ^2-statistic with 1 degree of freedom[2]:

$$X^2 \overset{\mathsf{def}}{=} \sum_{i \in \{AA, Aa, aa\}} \frac{(N_i - E_i)^2}{E_i} \ .$$

2.2 Linkage Disequilibrium

Another important notion in genetic association studies is linkage disequilibrium (LD). Linkage disequilibrium is an association in the genotypes at two loci in a genome. Suppose A, a are possible alleles at locus 1 and B, b are possible alleles at locus 2. In this case there are 9 possible genotypes: $AABB, AABb, AAbb, AaBB, AaBb, Aabb, aaBB, aaBb, aabb$. For $i, i' \in \{A, a\}$ and $j, j' \in \{B, b\}$ we use $N_{ii'jj'}$ to denote the observed count of genotype $ii'jj'$. As before, let N be the total size of the population sample:

$$N \overset{\mathsf{def}}{=} \sum_{\substack{i,i' \in \{A,a\} \\ j,j' \in \{B,b\}}} N_{ii'jj'}.$$

We consider the population frequencies of alleles A, a, B, b:

$$p_A \overset{\mathsf{def}}{=} \frac{\sum_{j,j' \in \{B,b\}} 2N_{AAjj'} + N_{Aajj'}}{2N} \ , \quad p_a \overset{\mathsf{def}}{=} \frac{\sum_{j,j' \in \{B,b\}} 2N_{aajj'} + N_{Aajj'}}{2N} \ ,$$

[2] 1 degree of freedom = 3 genotypes − 2 alleles

$$p_B \overset{\mathsf{def}}{=} \frac{\sum_{i,i'\in\{A,a\}} 2N_{ii'BB} + N_{ii'Bb}}{2N} \ , \qquad p_b \overset{\mathsf{def}}{=} \frac{\sum_{i,i'\in\{A,a\}} 2N_{ii'bb} + N_{ii'Bb}}{2N} \ .$$

Moreover, there are exactly 4 haplotypes to consider: AB, Ab, aB, ab. For $i \in \{A, a\}$ and $j \in \{B, b\}$, we use N_{ij} to denote the observed count for the haplotype ij and consider the population frequencies

$$p_{AB} \overset{\mathsf{def}}{=} \frac{N_{AB}}{2N} \ , \quad p_{Ab} \overset{\mathsf{def}}{=} \frac{N_{Ab}}{2N} \ , \quad p_{aB} \overset{\mathsf{def}}{=} \frac{N_{aB}}{2N} \ , \quad p_{ab} \overset{\mathsf{def}}{=} \frac{N_{ab}}{2N} \ .$$

Under *linkage equilibrium*, we expect the allele frequencies to be independent. In other words, we expect

$$p_{AB} = p_A p_B \ , \quad p_{Ab} = p_A p_b \ , \quad p_{aB} = p_a p_B \ , \quad p_{ab} = p_a p_b \ .$$

If the alleles are in *linkage disequilibrium*, the frequencies will deviate from the values above by a scalar D, so that

$$p_{AB} = p_A p_B + D \ , \quad p_{Ab} = p_A p_b - D \ , \quad p_{aB} = p_a p_B - D \ , \quad p_{ab} = p_a p_b + D \ .$$

The scalar D is easy to calculate: $D = p_{AB}p_{ab} - p_{Ab}p_{aB} = p_{AB} - p_A p_B$. However, the range of D depends on the frequencies, which makes it difficult to use it as a measure of disequilibrium. One of two scaled-down variants is used instead, the D'-measure or the r^2-measure.

D'-Measure. The D'-measure is defined as:

$$D' \overset{\mathsf{def}}{=} \frac{|D|}{D_{\mathsf{max}}} \quad \text{where} \quad D_{\mathsf{max}} = \begin{cases} \min\{p_A p_b, p_a p_B\} & \text{if } D > 0, \\ \min\{p_A p_B, p_a p_b\} & \text{if } D < 0. \end{cases}$$

r^2- Measure. The r^2 measure is given by

$$r^2 \overset{\mathsf{def}}{=} \frac{X^2}{N} \ , \quad \text{where} \quad X^2 \overset{\mathsf{def}}{=} \sum_{\substack{i\in\{A,a\} \\ j\in\{B,b\}}} \frac{(N_{ij} - E_{ij})^2}{E_{ij}} \ ,$$

where N_{ij} is the observed count and $E_{ij} \overset{\mathsf{def}}{=} N p_i p_j$ is the expected count. Using the fact that $|N_{ij} - E_{ij}| = ND$, it can be shown that

$$r^2 = \frac{D^2}{p_A p_B p_a p_b} \ .$$

The range of both D' and r^2 is $[0, 1]$. A value of 0 indicates perfect equilibrium and a value of 1 indicates perfect disequilibrium.

EM Algorithm for Haplotyping. Using the D' and r^2 LD measures described above requires knowing the observed haplotype counts or frequencies. However, haplotype counts (resp. frequencies) cannot be exactly determined from genotype counts (resp. frequencies). For example, consider 2 bi-allelic loci with alleles A, a and B, b. An observed genotype $AaBb$ can be one of two possible haplotypes: $(AB)(ab)$ or $(Ab)(aB)$. In practice, the Estimation Maximization (EM) algorithm can be used to *estimate* haplotype frequencies from genotype counts.

The EM algorithm starts with arbitrary initial values $p_{AB}^{(0)}, p_{Ab}^{(0)}, p_{aB}^{(0)}, p_{ab}^{(0)}$ for the haplotype frequencies, and iteratively updates them using the observed genotype counts. In each iteration, the current estimated haplotype frequencies are used in an *estimation step* to calculate the expected genotype frequencies (assuming the initial values are the true haplotype frequencies). Next, in a *maximization step*, these are used to estimate the haplotype frequencies for the next iteration. The algorithm stops when the haplotype frequencies have stabilized.

mTH ESTIMATION STEP

$$E^{(m)}_{AB/ab} \overset{\mathsf{def}}{=} \mathbb{E}\left[N_{AB/ab} \mid N_{AaBb}, p^{(0)}_{AB}, p^{(0)}_{Ab}, p^{(0)}_{aB}, p^{(0)}_{ab}\right] = N_{AaBb} \cdot \frac{p^{(m-1)}_{AB} p^{(m-1)}_{ab}}{p^{(m-1)}_{AB} p^{(m-1)}_{ab} + p^{(m-1)}_{Ab} p^{(m-1)}_{aB}}$$

$$E^{(m)}_{Ab/aB} \overset{\mathsf{def}}{=} \mathbb{E}\left[N_{Ab/aB} \mid N_{AaBb}, p^{(0)}_{AB}, p^{(0)}_{Ab}, p^{(0)}_{aB}, p^{(0)}_{ab}\right] = N_{AaBb} \cdot \frac{p^{(m-1)}_{Ab} p^{(m-1)}_{aB}}{p^{(m-1)}_{AB} p^{(m-1)}_{ab} + p^{(m-1)}_{Ab} p^{(m-1)}_{aB}}$$

mTH MAXIMIZATION STEP

$$\begin{aligned}
N^{(m)}_{AB} &= 2N_{AABB} + N_{AABb} + N_{AaBB} + E^{(m)}_{AB/ab} \\
N^{(m)}_{ab} &= 2N_{aabb} + N_{aaBb} + N_{Aabb} + E^{(m)}_{AB/ab} \\
N^{(m)}_{Ab} &= 2N_{AAbb} + N_{AABb} + N_{Aabb} + E^{(m)}_{Ab/aB} \\
N^{(m)}_{aB} &= 2N_{aaBB} + N_{AaBB} + N_{aaBb} + E^{(m)}_{Ab/aB}
\end{aligned}$$

2.3 Cochran-Armitage Test for Trend (CATT)

Finally, we consider the Cochran-Armitage Test for Trend (CATT), which is used in case-control studies to determine if a candidate allele is associated to a disease. We first describe the basic structure of case-control studies, and then describe the CATT test.

Case-Control Studies. As mentioned above, a case-control study is used to determine if a candidate allele A is associated to a specified disease. Such a study compares the genotypes of individuals who have the disease (cases) to the genotypes of individuals who do not (controls). A 2×3 contingency table of 3 genotypes vs. case/controls can be constructed with this information, as below, where the N_{ij} represent a number of individuals, R_i is the sum of the ith row, and C_j is the sum of the jth column. For example, N_{10} is the number of individuals with genotype AA who present the disease (affected phenotype), $R_0 = N_{00} + N_{01} + N_{02}$, $C_0 = N_{00} + N_{10}$, etc.

	AA	Aa	aa	Sum
Controls	N_{00}	N_{01}	N_{02}	R_0
Cases	N_{10}	N_{11}	N_{12}	R_1
Sum	C_0	C_1	C_2	N

Cochran-Armitage Test for Trend (CATT). Given a contingency table as above, the CATT computes the statistic

$$T \overset{\mathsf{def}}{=} \sum_{i=0}^{2} w_i (N_{0i} R_1 - N_{1i} R_0),$$

where $\vec{w} \overset{\mathsf{def}}{=} (w_0, w_1, w_2)$ is a vector of pre-determined weights[3], and the difference $(N_{0i}R_1 - N_{1i}R_0)$ can be thought of as the difference $N_{0i} - N_{1i}$ of controls and cases for a specific genotype, after reweighing the rows in the table to have the same sum.

[3]Common choices for the set of weights $\vec{w} = (w_0, w_1, w_2)$ are: $\vec{w} = (0, 1, 2)$ for the additive (co-dominant) model, $\vec{w} = (0, 1, 1)$ for the dominant model (A is dominant over a), and $\vec{w} = (0, 0, 1)$ for the recessive model (A is recessive to allele a).

The test statistic X^2 is defined to be

$$X^2 \overset{\mathsf{def}}{=} \frac{T^2}{\mathsf{Var}\,(T)}\ ,$$

where $\mathsf{Var}\,(T)$ is the variance of T:

$$\mathsf{Var}\,(T) = \frac{R_0 R_1}{N}\left(\sum_{i=0}^{2} w_i^2 C_i (N - C_i) - 2\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} w_i w_j C_i C_j\right).$$

To determine if a trend can be inferred, the CATT compares the test statistic X^2 to a χ^2-statistic with 1 degree of freedom.

2.4 Linear Regression

Linear regression is used in cases when the phenotype or trait is a continuous variable (e.g. tumor size) rather than a binary variable (e.g. whether a disease is present or not). It assumes a linear relationship between trait values and the genotype. The input data is a set of N pairs $(y_i, \vec{x}_i)$, where $y_i \in \{0, 1, 2\}$ is a genotype[4], and $\vec{x}_i \in \mathbb{R}^k$ is the vector of trait values corresponding to the individual with genotype x_i. Define

$$\mathbf{y} \overset{\mathsf{def}}{=} \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \quad , \quad \mathbf{X} \overset{\mathsf{def}}{=} \begin{pmatrix} \vec{x}_1^\top \\ \vdots \\ \vec{x}_N^\top \end{pmatrix}$$

Linear regressing finds $\boldsymbol{\beta} \in \mathbb{R}^k$ and $\boldsymbol{\varepsilon} \in \mathbb{R}^N$ such that $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$.

Linear regression models can be found using the least squares approach, and a solution to approximating least squares on homomorphically encrypted data is considered by Graepel, et al. [GLN13, Section 3.1]. Their work focuses on Fisher's linear discriminant classifier, but as noted there, linear regression can be cast in a similar framework. We refer the reader to their work for more details.

3 Practical Homomorphic Encryption

Fully homomorphic encryption (FHE) enables one to perform arbitrary computations on encrypted data, without first decrypting the data and without any knowledge of the secret decryption key. The result of the computation is given in encrypted form and can only be decrypted by a legitimate owner of the private decryption key. The first construction of FHE was shown by Gentry in 2009 [Gen09], and many improvements and new constructions have been presented in recent years [vDGHV10, BV11b, BV11a, BGV12, FV12, LTV12, Bra12, BLLN13, GSW13, BV14].

Model of Computation. In Gentry's initial work and many follow-up papers, computation is modeled as a boolean circuit with XOR and AND gates, or equivalently, as an arithmetic circuit over $\mathbb{F}_2$[5]. The data is encrypted bit-wise, which means that a separate ciphertext is produced for each bit in the message. Addition and multiplication operations are then performed on the

[4]For a bi-allelic gene with alleles A and a, the value 0 corresponds to the genotype AA, the value 1 corresponds to the genotype Aa and the value 2 corresponds to the genotype aa.

[5]An arithmetic circuit over $\mathbb{F}_t$ has addition and multiplication gates modulo t.

encrypted bits. Unfortunately, breaking down a computation into bit operations can quickly lead to a large and complex circuit, thus making homomorphic computation very inefficient.

Luckily, most known constructions allow the computation to take place over a larger message space. In particular, if the desired computation only needs to compute additions and multiplications of integer values (as is (almost)[6] the case with all algorithms presented in Section 2), then the data does not necessarily need to be expressed in a bitwise manner. Indeed, most known constructions allow the integers (or appropriate encodings of the integers) to be encrypted and homomorphic additions and multiplications to be performed over these integer values. The advantage of this approach is clear: a ciphertext now contains much more information than a single bit of data, making the homomorphic computation much more efficient.

It is important to note that in the latter approach, the only possible homomorphic operations are addition (equivalently, subtraction) and multiplication. It is currently not known how to perform division of integer values without performing an inefficient bitwise computation, as described above. For practical reasons, in this work we limit homomorphic operations to include only addition, subtraction, and multiplication.

Levels of Homomorphism. In all known FHE schemes, ciphertexts inherently contain a certain amount of noise, which "pollutes" the ciphertext. This noise grows during homomorphic operations and if the noise becomes too large, the ciphertext cannot be decrypted even with the correct decryption key. A *somewhat homomorphic encryption scheme* is one that can evaluate a limited number of operations (both addition and multiplication) before the noise grows large enough to cause decryption failures. Somewhat homomorphic schemes are usually very practical.

In order to perform an unlimited number of operations (and achieve *fully* homomorphic encryption), ciphertexts need to be constantly refreshed in order to reduce their noise. This is done using a costly procedure called *bootstrapping*.

A *leveled homomorphic encryption scheme* is one that allows the setting of parameters so as to be able to evaluate a given computation. In other words, given a fixed function that one wishes to compute, it is possible to select the parameters of the scheme in a way that allows one to homomorphically compute the specified function, without the use of the costly bootstrapping step. Leveled homomorphic schemes enjoy the flexibility of fully homomorphic schemes, in that they can homomorphically evaluate any function, and are also quite practical (albeit not as practical as somewhat homomorphic schemes). The construction we use in our implementation is a leveled homomorphic encryption scheme.

3.1 The Homomorphic Encryption Scheme

In our implementation we use a modified[7] version of the homomorphic encryption scheme proposed by López-Alt and Naehrig [LN14b], which is based on the schemes [SS11, LTV12, Bra12, BLLN13]. The scheme is a public-key encryption scheme and consists of the following algorithms:

- A key generation algorithm $\mathsf{KeyGen}(\mathsf{params})$ that, on input the system parameters params, generates a public/private key pair $(\mathsf{pk}, \mathsf{sk})$.
- An encryption algorithm $\mathsf{Encrypt}(\mathsf{pk}, m)$ that encrypts a message m using the public key pk.

[6]The algorithms in Section 2 include divisions. In Section 4, we show how to get around this issue.

[7]The only modification we make to the scheme of López-Alt and Naehrig is removing a step called "relinearization" or "key switching", needed to make decryption independent of the function that was homomorphically evaluated. In our implementation, decryption depends on the number of homomorphic multiplications that were performed. We make this change for efficiency reasons, as relinearization is very costly.

- A decryption algorithm $\mathsf{Decrypt}(\mathsf{sk}, c)$ that decrypts a ciphertext c with the private key sk.
- A homomorphic addition function $\mathsf{Add}(c_1, c_2)$ that given encryptions c_1 and c_2 of m_1 and m_2, respectively, outputs a ciphertext encrypting the sum $m_1 + m_2$.
- A homomorphic multiplication function $\mathsf{Mult}(c_1, c_2)$ that, given encryptions c_1 and c_2 of m_1 and m_2, respectively, outputs a ciphertext encrypting the product $m_1 m_2$.

System Parameters. The scheme operates in the ring $R \overset{\mathsf{def}}{=} \mathbb{Z}[X]/(X^n + 1)$, whose elements are polynomials with integer coefficients of degree less than n. All messages, ciphertexts, encryption and decryption keys, etc. are elements in the ring R, and have this form. In more detail, an element $a \in R$ has the form $a = \sum_{i=0}^{n-1} a_i X^i$, with $a_i \in \mathbb{Z}$. Addition in R is done component-wise in the coefficients, and multiplication is simply polynomial multiplication modulo $X^n + 1$.

The scheme also uses an integer modulus q. In what follows, we use the notation $[a]_q$ to denote the operation of reducing the coefficients of $a \in R$ modulo q into the set $\{-\lfloor \frac{q}{2} \rfloor, \ldots, \lfloor \frac{q}{2} \rfloor\}$.

Finally, the scheme uses two probability distributions on R, χ_{key} and χ_{err}, which generate polynomials in R with small coefficients. In our implementation, we let the distribution χ_{key} be the uniform distribution on polynomials with coefficients in $\{-1, 0, 1\}$. Sampling an element according to this distribution means sampling all its coefficients uniformly from $\{-1, 0, 1\}$. For the distribution χ_{err}, we use a discrete Gaussian distribution with mean 0 and appropriately chosen standard deviation (see Section 4.4). For clarity of presentation, we refrain from formally describing the specifics of this distribution and instead refer the reader to any of [SS11, LTV12, Bra12, BLLN13, LN14b] for a formal definition.

The system parameters of the scheme are the degree n, modulus q, and distributions $\chi_{\text{key}}, \chi_{\text{err}}$: $\mathsf{params} = (n, q, \chi_{\text{key}}, \chi_{\text{err}})$.

Plaintext Space. The plaintext space of the scheme is the set of integers in the interval $\mathcal{M} = [-2^n, 2^n]$. For the scheme to work correctly, we assume that the initial inputs, the output of the function evaluation, and all intermediate values are all in $\mathcal{M}$.

To encrypt an integer $\mu \in \mathcal{M}$, this integer is first encoded as a polynomial $m \in R$. To do this, we take the bit-decomposition of μ and use these bits as the coefficients in m. Formally, if $\mu = \sum_i \mu_i 2^i$ for $\mu_i \in \{0, 1\}$, then we define $m = \sum_i \mu_i X^i$.

Formal Definition. Below is a formal and detailed definition of the key generation, encryption, decryption, and homomorphic evaluation algorithms.

- $\mathsf{KeyGen}(\mathsf{params})$: On input the parameters $\mathsf{params} = (n, q, \chi_{\text{key}}, \chi_{\text{err}})$, the key generation algorithm samples polynomials $f', g \leftarrow \chi_{\text{key}}$ from the key distribution and sets

$$f = [(X-2)f' + 1]_q.$$

  If f is not invertible modulo q, it chooses a new f'. Otherwise, it computes the inverse f^{-1} of f in R modulo q. Finally, it outputs the key pair:

$$\mathsf{pk} = h \overset{\mathsf{def}}{=} [g f^{-1}]_q \in R \quad \text{and} \quad \mathsf{sk} = f \in R.$$

- $\mathsf{Encrypt}(h, \mu)$: To encrypt an integer $\mu \in \mathcal{M}$, the encryption algorithm first encodes it as a polynomial m, as described above. Then, it samples small error polynomials $s, e \leftarrow \chi_{\text{err}}$, and outputs the ciphertext
$$c \overset{\mathsf{def}}{=} [\Delta m + e + hs]_q \in R,$$
where
$$\Delta \overset{\mathsf{def}}{=} \lceil q\Upsilon \rceil \quad \text{and} \quad \Upsilon \overset{\mathsf{def}}{=} -\frac{X^{n-1} + 2X^{n-2} + 4X^{n-3} + \ldots + 2^{n-1}}{2^n + 1} \in \mathbb{Q}[X].$$

- $\mathsf{Add}(c_1, c_2)$: Given two ciphertexts c_1 and c_2, outputs the ciphertext $c_{\mathsf{add}} \overset{\mathsf{def}}{=} [c_1 + c_2]_q$.

- $\mathsf{Mult}(c_1, c_2)$: Given two ciphertexts c_1 and c_2, outputs $c_{\mathsf{mult}} \overset{\mathsf{def}}{=} [c_1 c_2]_q$.

- $\mathsf{Decrypt}(f, c)$: Given the private decryption key f and a ciphertext c that is the output of a degree-D function evaluation[8], the decryption algorithm computes $\widetilde{f} \overset{\mathsf{def}}{=} f^D \in R$ and
$$\mu = \left(\left\lfloor \frac{(X-2)}{q} \cdot [\widetilde{f}c]_q \right\rceil \mod (X-2) \right) \mod 2^n + 1.$$

  We remark that if the function that will be homomorphically computed is known in advance (or even only its degree), then the polynomial f^D can be precomputed when the secret key is generated, simplifying the decryption step to a single polynomial multiplication and some modular operations.

  We also note that the modular reduction modulo $(X-2)$ is mathematically equivalent to the evaluation of the polynomial at the point $X = 2$.

4 Computation on Encrypted Data

In this section, we discuss how to run the statistical algorithms described in Section 2 on genetic data encrypted using the homomorphic encryption scheme described in Section 3. To this end, in Section 4.1 we describe how genetic data can be encoded and encrypted. In Section 4.2 we discuss how to obtain the genotype and phenotype frequencies that serve as input to the algorithms described in Section 2. Additionally, given the constraints of homomorphic computation on encrypted data, we must make some necessary modifications to the statistical algorithms; we describe these in Section 4.3. Finally, in Section 4.4, we discuss how to choose the parameters of the encryption scheme. In what follows, for a value a, we use $\widehat{a}$ to denote an encryption of a.

4.1 Encoding Genomic Data

Structure of the Data. Data used in genetic association studies consists of individuals' genotypes and phenotypes. The data can be represented in 2 tables or matrices, one for genotype information and the other for phenotype information. In the genotypes table, each row contains information about a single person, and each column specifies a DNA locus. An entry in this table specifies the person's genotype at the given locus. For a bi-allelic gene with alleles A, a, this can be one of 4 possible values: the reference homozygote AA (value 0), the heterozygote Aa (value 1),

[8]Informally, a function has degree D if it can be represented as a (possibly multivariate) polynomial of degree D. See Section 4.4 for more details.

the non-reference homozygote aa (value 2) and "missing" if that person's genotype at the specified locus is not known.

Similarly, in the phenotypes table, each row contains information about a single person, and each column specifies a single phenotype. An entry in this table specifies the person's given phenotype. For a disease phenotype, this can be one of 3 possible values: unaffected (value 0), affected (value 1) and "missing" if that person's affection status is not known. For continuous phenotypes (e.g. tumor size), the table entry contains a real number. We focus only on phenotypes containing disease affection status.

Genotype Encoding. For each entry (i, j) in the genotype table, we compute 3 ciphertexts, one for each of the possible values 0,1,2 (ie. AA, Aa, aa); we call these ciphertexts $c_0^{(i,j)}, c_1^{(i,j)}, c_2^{(i,j)}$ respectively. A ciphertext encrypts 1 if the entry value is the same as the value it represents, and 0 otherwise. More specifically, the 4 possible genotypes are encoded as follows:

$$\begin{aligned}
AA \text{ (value 0)} : \quad & c_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1), \quad c_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad c_2^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \\
Aa \text{ (value 1)} : \quad & c_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad c_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1), \quad c_2^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \\
aa \text{ (value 2)} : \quad & c_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad c_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad c_2^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1), \\
\text{missing} : \quad & c_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad c_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad c_2^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0).
\end{aligned}$$

Phenotype Encoding. For each entry (i, j) in the phenotype table, we compute 2 ciphertexts, one for the "unaffected" phenotype (value 0) and one for the"affected" phenotype (value 1); we call these ciphertexts $c_0^{(i,j)}, c_1^{(i,j)}$ respectively. A ciphertext encrypts 1 if the entry value is the same as the value it represents, and 0 otherwise. More specifically, the 3 possible genotypes are encoded as follows:

$$\begin{aligned}
\text{unaffected (value 0)} : \quad & z_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1), \quad z_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \\
\text{affected (value 1)} : \quad & z_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad z_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 1), \\
\text{missing} : \quad & z_0^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0), \quad z_1^{(i,j)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0).
\end{aligned}$$

4.2 Computing Genotype and Phenotype Counts

Recall that the statistical algorithms described in Section 2 take as input genotype and phenotype frequencies or counts. While we are not able to obtain the genotype and phenotype *frequencies*[9], we can obtain the *counts* using a few simple homomorphic additions. Indeed, if the data is encrypted as described in Section 4.1, computing the (encrypted) counts $\widehat{N}_k^{(j)}$ of value-k genotypes at locus j can be done by summing all the ciphertexts $c_k^{(i,j)}$ in column j of the genotype table:

$$\widehat{N}_0^{(j)} = \sum_i c_0^{(i,j)}, \quad \widehat{N}_1^{(j)} = \sum_i c_1^{(i,j)}, \quad \widehat{N}_2^{(j)} = \sum_i c_2^{(i,j)}.$$

Finally, we can compute the (encrypted) total number $\widehat{N}^{(j)}$ of available (non-missing genotypes) in column j by summing

$$\widehat{N}^{(j)} = \widehat{N}_0^{(j)} + \widehat{N}_1^{(j)} + \widehat{N}_2^{(j)}.$$

[9]Recall from Section 3 that we cannot perform homomorphic divisions.

4.3 Modified Algorithms

Unfortunately, since we are not able to compute the genotype and phenotype *frequencies*, we must modify the statistical algorithms to use genotype and phenotype *counts* instead.

Pearson Goodness-of-Fit or Chi-Squared Test. Recall that for a single locus, the Pearson test computes the test statistic

$$X^2 = \sum_{i=0}^{2} \frac{(N_i - E_i)}{E_i},$$

where N_i is the observed genotype count, and E_i is the expected genotype count. The expected counts can be computed as

$$E_0 = N\left(\frac{2N_0 + N_1}{2N}\right)^2, \quad E_1 = 2N\left(\frac{2N_0 + N_1}{2N}\right)\left(\frac{2N_2 + N_1}{2N}\right), \quad E_2 = N\left(\frac{2N_2 + N_1}{2N}\right)^2,$$

which can be simplified to

$$E_0 = \frac{(2N_0 + N_1)^2}{4N}, \quad E_1 = \frac{(2N_0 + N_1)(2N_2 + N_1)}{2N}, \quad E_2 = \frac{(2N_2 + N_1)^2}{4N}.$$

The test statistic X^2 can then be computed as

$$\begin{aligned} X^2 &= \frac{(N_0 - E_0)^2}{E_0} + \frac{(N_1 - E_1)^2}{E_1} + \frac{(N_2 - E_2)^2}{E_2} \\ &= \frac{(4N_0N_2 - N_1^2)^2}{2N}\left(\frac{1}{2(2N_0 + N_1)^2} + \frac{1}{(2N_0 + N_1)(2N_2 + N_1)} + \frac{1}{2(2N_2 + N_1)^2}\right). \end{aligned}$$

Since we are unable to perform homomorphic divisions, we return encryptions of $\alpha, N, \beta_1, \beta_2, \beta_3$, where

$$\alpha \overset{\text{def}}{=} (4N_0N_2 - N_1^2)^2, \ \beta_1 \overset{\text{def}}{=} 2(2N_0 + N_1)^2, \ \beta_2 \overset{\text{def}}{=} (2N_0 + N_1)(2N_2 + N_1), \ \beta_3 \overset{\text{def}}{=} 2(2N_2 + N_1)^2.$$

From these, the test statistic can be computed as:

$$X^2 = \frac{\alpha}{2N}\left(\frac{1}{\beta_1} + \frac{1}{\beta_2} + \frac{1}{\beta_3}\right).$$

EM Algorithm. To run the EM algorithm on genotypes at loci j and ℓ, we need the 9 genotype counts $N_{xy}^{(j,\ell)}$ for $x, y \in \{0, 1, 2\}$. In other words, we need to know the number of individuals $N_{x,y}^{(j,\ell)}$ in the data set that have genotype x at locus j and genotype y at locus ℓ, for all combinations of x and y. The (encrypted) counts can be computed as

$$\widehat{N}_{xy}^{(j,\ell)} = \sum_i c_x^{(i,j)} \cdot c_y^{(i,\ell)}.$$

Recall that the EM algorithm estimates the haplotype *frequencies*. As before, we are unable to estimate frequencies since we cannot perform homomorphic division, but we are able to estimate haplotype *counts*. Notice that since $N_{xy} = 2N \cdot p_{xy}$, this does not change the fraction in $\mu_{AB/ab}^{(m)}$ and $\mu_{Ab/aB}^{(m)}$ (essentially, this change multiplies both the numerator and the denominator by $4N^2$). This modifies the estimation step as follows.

*m*th Estimation Step

$$E^{(m)}_{AB/ab} = N_{11} \cdot \frac{N^{(m-1)}_{AB} N^{(m-1)}_{ab}}{N^{(m-1)}_{AB} N^{(m-1)}_{ab} + N^{(m-1)}_{Ab} N^{(m-1)}_{aB}} \overset{\mathsf{def}}{=} \frac{\alpha^{(m)}}{\beta^{(i)}},$$

$$E^{(m)}_{Ab/aB} = N_{11} \cdot \frac{N^{(m-1)}_{Ab} N^{(m-1)}_{aB}}{N^{(m-1)}_{AB} N^{(m-1)}_{ab} + N^{(m-1)}_{Ab} N^{(m-1)}_{aB}} \overset{\mathsf{def}}{=} \frac{\gamma^{(m)}}{\beta^{(i)}}.$$

We can also simplify the iteration so that at any given point, we need only remember one numerator and one denominator. Define

$$\zeta_{AB} \overset{\mathsf{def}}{=} 2N_{22} + N_{21} + N_{12}, \qquad \zeta_{ab} \overset{\mathsf{def}}{=} 2N_{00} + N_{01} + N_{10},$$
$$\zeta_{Ab} \overset{\mathsf{def}}{=} 2N_{20} + N_{21} + N_{10}, \qquad \zeta_{aB} \overset{\mathsf{def}}{=} 2N_{02} + N_{12} + N_{01}.$$

Then

$$\begin{aligned}
N^{(m)}_{AB} &= \zeta_{AB} + E^{(m)}_{AB/ab} = \zeta_{AB} + \frac{\alpha^{(m)}}{\beta^{(m)}} = \frac{\zeta_{AB} \cdot \beta^{(m)} + \alpha^{(m)}}{\beta^{(m)}},\\
N^{(m)}_{ab} &= \zeta_{ab} + E^{(m)}_{AB/ab} = \zeta_{ab} + \frac{\alpha^{(m)}}{\beta^{(m)}} = \frac{\zeta_{ab} \cdot \beta^{(m)} + \alpha^{(m)}}{\beta^{(m)}},\\
N^{(m)}_{Ab} &= \zeta_{Ab} + E^{(m)}_{Ab/aB} = \zeta_{Ab} + \frac{\gamma^{(m)}}{\beta^{(m)}} = \frac{\zeta_{Ab} \cdot \beta^{(m)} + \gamma^{(m)}}{\beta^{(m)}},\\
N^{(m)}_{aB} &= \zeta_{aB} + E^{(m)}_{Ab/aB} = \zeta_{aB} + \frac{\gamma^{(m)}}{\beta^{(m)}} = \frac{\zeta_{aB} \cdot \beta^{(m)} + \gamma^{(m)}}{\beta^{(m)}}.
\end{aligned}$$

Following the iteration, at the next estimation step we need to compute:

$$\begin{aligned}
E^{(m+1)}_{AB/ab} &= N_{11} \cdot \frac{\left(\frac{\zeta_{AB}\cdot\beta^{(m)}+\alpha^{(m)}}{\beta^{(m)}}\right)\left(\frac{\zeta_{ab}\cdot\beta^{(m)}+\alpha^{(m)}}{\beta^{(m)}}\right)}{\left(\frac{\zeta_{AB}\cdot\beta^{(m)}+\alpha^{(m)}}{\beta^{(m)}}\right)\left(\frac{\zeta_{ab}\cdot\beta^{(m)}+\alpha^{(m)}}{\beta^{(m)}}\right)+\left(\frac{\zeta_{Ab}\cdot\beta^{(m)}+\gamma^{(m)}}{\beta^{(m)}}\right)\left(\frac{\zeta_{aB}\cdot\beta^{(m)}+\gamma^{(m)}}{\beta^{(m)}}\right)}\\
&= N_{11} \cdot \frac{\left(\zeta_{AB} \cdot \beta^{(m)} + \alpha^{(m)}\right)\left(\zeta_{ab} \cdot \beta^{(m)} + \alpha^{(m)}\right)}{\left(\zeta_{AB} \cdot \beta^{(m)} + \alpha^{(m)}\right)\left(\zeta_{ab} \cdot \beta^{(m)} + \alpha^{(m)}\right) + \left(\zeta_{Ab} \cdot \beta^{(m)} + \gamma^{(m)}\right)\left(\zeta_{aB} \cdot \beta^{(m)} + \gamma^{(m)}\right)}\\
&\overset{\mathsf{def}}{=} \frac{\alpha^{(m+1)}}{\beta^{(m+1)}}.
\end{aligned}$$

and similarly,

$$\begin{aligned}
E^{(m+1)}_{Ab/aB} &= N_{11} \cdot \frac{\left(\zeta_{Ab} \cdot \beta^{(m)} + \gamma^{(m)}\right)\left(\zeta_{aB} \cdot \beta^{(m)} + \gamma^{(m)}\right)}{\left(\zeta_{AB} \cdot \beta^{(m)} + \alpha^{(m)}\right)\left(\zeta_{ab} \cdot \beta^{(m)} + \alpha^{(m)}\right) + \left(\zeta_{Ab} \cdot \beta^{(m)} + \gamma^{(m)}\right)\left(\zeta_{aB} \cdot \beta^{(m)} + \gamma^{(m)}\right)}\\
&\overset{\mathsf{def}}{=} \frac{\gamma^{(m+1)}}{\beta^{(m+1)}}.
\end{aligned}$$

In other words, since the denominator $\beta^{(m)}$ always cancels out, we need only remember the numerators. The numerators depend on $\beta^{(m)}$, so we still compute it as part of the numerator computation, but do not need to store it after this computation. Of course, at the last step we must divide by $\beta^{(m)}$ to maintain correctness.

The modified estimation and maximization steps are described below.

mTH ESTIMATION STEP

$$\alpha^{(m)} = N_{11} \cdot N_{AB}^{(m-1)} N_{ab}^{(m-1)}, \quad \gamma^{(m)} = N_{11} \cdot N_{Ab}^{(m-1)} N_{aB}^{(m-1)}, \quad \beta^{(m)} = N_{AB}^{(m-1)} N_{ab}^{(m-1)} + N_{Ab}^{(m-1)} N_{aB}^{(m-1)}.$$

mTH MAXIMIZATION STEP

$$N_{AB}^{(m)} = \zeta_{AB} \cdot \beta^{(m)} + \alpha^{(m)}, \; N_{ab}^{(m)} = \zeta_{ab} \cdot \beta^{(m)} + \alpha^{(m)}, \; N_{Ab}^{(m)} = \zeta_{Ab} \cdot \beta^{(m)} + \gamma^{(m)}, \; N_{aB}^{(m)} = \zeta_{aB} \cdot \beta^{(m)} + \gamma^{(m)}.$$

Measures of LD. The degree of linkage disequilibrium of two bi-allelic genes can be determined by computing the scalar value $D \overset{\mathsf{def}}{=} p_{AB} - p_A p_B$ where A and B are the reference alleles of the genes, p_A and p_B are their corresponding population frequencies, and p_{AB} is the population frequency of the haplotype AB. Once more, we need to compute D as a function of counts rather than frequencies, as we cannot compute the latter homomorphically. We have

$$D = \frac{N_{AB}}{2N} - \frac{N_A}{2N} \cdot \frac{N_B}{2N} = \frac{2N \cdot N_{AB} - N_A N_B}{(2N)^2}.$$

The haplotype count N_{AB} is estimated using the EM algorithm, which outputs values α, β such that $N_{AB} = \alpha/\beta$. Thus,

$$D = \frac{2N \cdot \alpha - \beta N_A N_B}{\beta(2N)^2} = \frac{2N \cdot \alpha - \beta(2N_{AA} + N_{Aa})(2N_{BB} + N_{Bb})}{\beta(2N)^2}.$$

Again, since we cannot perform homomorphic division, we return encryptions of δ and β, where

$$\delta \overset{\mathsf{def}}{=} 2N \cdot \alpha - \beta(2N_{AA} + N_{Aa})(2N_{BB} + N_{Bb}).$$

The scalar D can be computed as $D = \delta/(\beta(2N)^2)$. To be able to calculate the D' and r^2 statistics, we also return encryptions of N_A, N_a, N_B, N_b, from which they can be computed:

$$D' = \frac{\delta}{\beta D_{\mathsf{max}}}, \quad \text{where} \;\; D_{\mathsf{max}} = \begin{cases} \min\{N_A N_b, N_a N_B\} & \text{if } D > 0, \\ \min\{N_A N_B, N_a N_b\} & \text{if } D < 0 \end{cases}$$

and

$$r^2 = \frac{\delta^2}{\beta^2 N_A N_a N_B N_b}.$$

Cochran-Armitage Test for Trend. To run the CATT algorithm on genotype at locus j and phenotype ℓ, we need the 6 genotype-phenotype counts $N_{x,y}^{(j,\ell)}$ for $x \in \{0, 1, 2\}$ and $y \in \{0, 1\}$. In other words, we need to know the number of individuals $N_{x,y}^{(j,\ell)}$ in the data set that have genotype x at locus j and phenotype ℓ with value y, for all combinations of x and y. The (encrypted) counts can be computed as

$$\widehat{N}_{xy}^{(j,\ell)} = \sum_i c_x^{(i,j)} \cdot z_y^{(i,\ell)}.$$

The test statistic can be computed as $X^2 = \alpha/\beta$ where

$$\alpha \overset{\mathsf{def}}{=} N \cdot \left(\sum_{i=0}^{2} w_i (N_{0i} R_1 - N_{1i} R_0) \right)^2, \quad \beta \overset{\mathsf{def}}{=} R_0 R_1 \cdot \left(\sum_{i=0}^{2} w_i^2 C_i (N - C_i) - 2 \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_i w_j C_i C_j \right).$$

4.4 How to Set Parameters

In order to implement any cryptographic scheme efficiently and securely, one needs to select suitable parameters. For the homomorphic encryption scheme in this work, one needs to find a dimension n, an integer modulus q and the standard deviation σ of the error distribution χ_{err}. These parameters have to satisfy two conditions. The first one guarantees security, more precisely the parameters need to guarantee a desired level λ of security against known attacks on the homomorphic encryption scheme. This means that an attacker is forced to run for at least 2^λ steps in order to break the scheme. The second condition guarantees correctness. Given a desired computation, the encryption scheme must be able to correctly evaluate the computation without the error terms in the ciphertexts growing too large such that the result is decrypted correctly. Subject to these two conditions, we aim to choose the smallest dimension and modulus in order to make computations as efficient as possible. We follow the approach described in [LN14a] for selecting parameters and refer the reader to this paper for more details.

Security. One first picks a desired security level, common values are $\lambda = 80$ or higher. Next, one chooses a dimension n and standard deviation σ. The analysis in [LN14a] shows that, for fixed λ, n and σ, there is an upper bound on the allowed modulus q to achieve the desired security level. In order to increase security, one can either increase the dimension n, or increase the standard deviation σ or a combination of both and then re-evaluate to obtain the new maximal value for q.

Correctness. The correctness condition is in contrast to the security condition in that given a dimension and standard deviation, it demands a lower bound for the modulus q. The complexity of the planned computation and the size of σ influence the error growth throughout the computation. To make correct decryption possible, the relative size of the error compared to the modulus q needs to be small enough, or in other words, q needs to be large enough to accommodate the error that arises during the homomorphic operations.

Efficiency. To maximize the efficiency of our implementation, we select the "smallest" parameter set amongst those that satisfy the security and correctness criteria. Clearly, increasing n or q leads to a decrease in efficiency, so we are interested to keep the dimension and modulus as small as possible. In general, smaller security level and less complex computations allow for smaller parameters, increasing the security and complexity leads to larger, less efficient parameters. In this work, we are contented with a security level of $\lambda = 80$.

5 Performance

In this section, we describe our experiments. We implemented the homomorphic encryption scheme from Section 3 and the algorithms described in Section 4 in the computer algebra system Magma [BCP97]. Note that specialized implementations in a language such as C/C++ may perform significantly better than our proof-of-concept implementation in Magma. The exact speed-up depends on the optimizations in such an implementation. For example, for the parameters used in [BLLN13], we observe that our Magma implementation of the homomorphic addition, multiplication, and decryption operations is roughly twice as slow as the C/C++ implementation reported in [LN14a], which uses a general purpose C/C++ library for the underlying arithmetic. The decryption operation in the implementation in [LN14a] in turn is roughly twice as slow as the C implementation in [BLLN13]. A completely specialized and optimized implementation will achieve even better efficiency.

Timings for the Scheme. Timings for the basic algorithms of the homomorphic encryption scheme (key generation, encryption, addition, multiplication, and decryption for several degree values D) are shown in Table 1. Timings for both key generation and encryption include the sampling of small elements according to the distributions described in Section 3. Depending on the specific implementation scenario, these steps could be precomputed and their cost amortized.

As mentioned in Section 3, our implementation does not perform relinearization (a.k.a. key-switching) in homomorphic operations (see any of [BV11a, BGV12, LTV12, Bra12, BLLN13, LN14b]). We choose not to perform this step as an optimization (indeed, the timing for a single multiplication increased more than 50-fold when relinearization was included). The downside to our approach is that decryption depends on the degree of the function that was homomorphically computed (recall from Section 3 that the decryption algorithm first computes f^D where f is the secret key and D is the degree of the computed function). Thus, decryption timings depend on the degree of the evaluated function, albeit only logarithmically. We remark that if the function is known in advance (or even only an upper bound on its degree), the element f^D can be precomputed. In this case, the decryption time in *all* cases is the same and equivalent to the decryption time for degree-1 ciphertexts.

Parameters. Table 1 provides timings for two different parameter sets. The first set (I) uses smaller parameters and therefore produces faster timings. All algorithms in this paper can be run correctly with the first parameter set except for the EM algorithm for more than two iterations. The second set (II) uses larger parameters that are suitable to run the EM algorithm for 3 iterations, but the performance is worse due to the larger parameters. Both parameter sets provide 80 bits of security. We refer the reader to Section 4.4 for a detailed explanation of how these parameter sets were selected.

In order to increase the security to 128 bits, we must adjust the parameter sizes. For example, this can be done as follows. According to the analysis in Section 4.4, when all other parameters are fixed, one can achieve the 128-bit security level by decreasing the modulus q to 149 bits. Such a parameter set can still be used to run the same algorithms as parameter set (I), except for the LD algorithm. In order to run the LD algorithm, one needs to increase the dimension n. If n is restricted to be a power of two, then $n = 8192$ as in parameter set (II). However, q needs to be smaller than in set (II). Arithmetic for such parameters is the same as for the set (II) but with slightly faster arithmetic modulo q. Therefore, the timings in Table 1 give a rough estimate for the upper bound on the performance penalty when moving to 128-bit security.

Table 1: Timings for the operations of the homomorphic encryption scheme. Measurements were done in the computer algebra system Magma [BCP97] V2.17-8 on an Intel(R) Core(TM) i7-3770S CPU @ 3.10GHz, 8GB RAM, running 64-bit Windows 8.1. Values are the mean of 1000 measurements of the respective operation. Decryption depends on the degree of the evaluated function, the timing differences are due to the computation of the respective power of the secret key. Parameter sets are (I) $n = 4096$, $\lceil \log(q) \rceil = 192$ and (II) $n = 8192$, $\lceil \log(q) \rceil = 384$, both use $\sigma = 8$ and provide 80-bit security. A single ciphertext with parameter set (I) is of size slightly less than 100KB, for parameter set (II), it is less than 400KB.

Operation	KeyGen	Encrypt	Add	Mult	Decrypt				
					deg 1	deg 2	deg 5	deg 10	deg 20
Parameters I	3.599s	0.296s	0.001s	0.051s	0.035s	0.064s	0.114s	0.140s	0.164s
Parameters II	18.141s	0.783s	0.003s	0.242s	0.257s	0.308s	0.598s	0.735s	0.888s

Testing Correctness. To test the correctness of our homomorphic evaluations, we implemented the statistical algorithms in their original form (as described in Section 2) and unencrypted, as well as the modified algorithms described in Section 4, also unencrypted. A third implementation ran the modified algorithms (as in Section 4) on encrypted data and used the homomorphic operations of the encryption scheme. In each test, we ran all versions of the algorithms and confirmed that their return values were equal.

Data Pre-Processing. All algorithms being considered take as input genotype and/or phenotype count tables. Because of this, once the encrypted tables have been computed and appropriate parameters have been chosen, the running times of the statistical algorithms are independent of the size of the population sample and depend only on the parameter set needed for the computation.[10] Thus, we separate our analysis into two phases: In the first phase, we construct the encrypted genotype and phenotype tables. This includes encoding and encrypting genotype and phenotype data, as well as summing these encryptions (see Section 4.1 and Section 4.2). In the second phase, we run the statistical algorithms on the encrypted tables. Indeed, we view the first phase as a pre-processing of the data, whose cost can be amortized over any number of computations. Moreover, this data can be easily updated by subtracting encryptions if a person's data is no longer needed or desired, by adding new encryptions if new data is collected, or by replacing specific encryptions as a person's record needs to be updated or modified. We emphasize the fact that there is no need to re-encode and re-encrypt the entire data set when modifications are required. Necessary changes will be proportional to the number of entries that need to be modified (inserted, deleted, or updated).

The main cost in pre-processing the data is the computation of the 3 encryptions for each genotype sample and the 2 encryptions for each phenotype sample (see Section 4.1). This cost is linear in the size of the data set and can be easily computed from the timings for encryption given in Table 1. For example, encoding and encrypting 1000 genotype data points sequentially using parameter set (I) takes roughly 15 minutes, and encoding and encrypting 1000 phenotype data entries takes roughly 10 minutes.

Once all genotypes and phenotypes have been encoded and encrypted, we need to construct 3 contingency tables (see Section 4.2 and Section 4.3). The first table contains the genotype counts for a single locus and can be computed by sequential addition of the genotype encryptions. Sequentially adding 1000 ciphertexts takes roughly 1 second; thus, computing all genotype counts for a single locus takes roughly 3 seconds. Computing the 3×3 contingency table for the counts of individuals having a certain genotype at one locus and another at a second locus requires one multiplication and one addition per individual. Thus, for parameter set (I), each entry in the table can be computed in roughly 1 minute and the entire table can be computed in roughly 9 minutes. Similarly, computing the 2×3 contingency table for the counts of individuals with a certain genotype and a given phenotype requires one multiplication and one addition per individual. Thus, for parameter set (I), the entire table can be computed in roughly 6 minutes.

[10] Admittedly, the size of the parameters needed does depend on the magnitude of the genotype and phenotype counts, which can be as large as the size of the population sample. This is because the size of the message encrypted at any given time (i.e. the size of the counts and all the intermediate values in the computation) cannot grow too large relative to the modulus q. Therefore, larger population sizes (and therefore larger counts) require a larger modulus q, which in turn requires a larger dimension n for security. However, for a fixed parameter set, it is possible to compute an upper bound on the size of the population sample and the homomorphic computations detailed in this work do work correctly for any population sample with size smaller than the given bound.

Timings for the Statistical Algorithms. As mentioned above, once the data has been processed and the genotype and phenotype tables have been computed, the runtime of the statistical algorithms is independent of the size of the population sample and only depends on the parameter set needed for the computation. Table 2 contains performance numbers for the algorithms after the data has been encoded and encrypted, and population counts have been computed. It includes timings for both parameter sets described above.

Table 2: Timings for statistical algorithms. Measurements were done in the computer algebra system Magma [BCP97] V2.17-8 on an Intel(R) Core(TM) i7-3770S CPU @ 3.10GHz, 8GB RAM, running 64-bit Windows 8.1. Values are the mean of 100 measurements of the respective algorithm.

Algorithm	Pearson	EM			LD	CATT
		1 iteration	2 iterations	3 iterations		
Parameters I	0.34s	0.57s	1.10s	–	0.19s	0.94s
Parameters II	1.36s	2.29s	4.54s	6.85s	0.74s	3.63s

Further Specialization. For the case that only one of the statistical algorithms needs to be run, further optimizations are possible, decreasing storage space and runtime significantly. For example, if we focus on running only the Pearson test, we can change the encoding of genotypes from Section 4.1 to use only a single ciphertext as follows: value 0 is encrypted as $c^{i,j} = \mathsf{Encrypt}(\mathsf{pk}, 1)$, value 1 as $c^{i,j} = \mathsf{Encrypt}(\mathsf{pk}, x^{100})$, value 2 as $c^{i,j} = \mathsf{Encrypt}(\mathsf{pk}, x^{301})$ and missing values as $c^{i,j} = \mathsf{Encrypt}(\mathsf{pk}, 0)$. By adding up all such ciphertexts, the genotype counts are then contained in a single ciphertexts that encrypts $N_0 + N_1x^{100} + N_2x^{301}$. The Pearson test has degree 4 in these counts and can be computed with only two multiplication operations on this ciphertext. Note that needed values encoded in the polynomials $(N_0 + N_1x^{100} + N_2x^{301})^i$ for $i \in \{2, 4\}$ can be shifted to the constant coefficient by multiplying with suitable powers of x.

Using this optimization, the storage space for encrypted genotype data is reduced by a factor 3, as is the encryption time. With parameter set (I), the runtime of one Pearson test becomes less than 0.13s.

6 Conclusion and future work

In this paper we presented algorithms and proof-of-concept implementations for computing on encrypted genomic data. We showed how to encode genotype and phenotype data for encryption and how to apply the Pearson Goodness-of-Fit test, the D' and r^2-measures of linkage disequilibrium, the Estimation Maximization (EM) algorithm for haplotyping, and the Cochran-Armitage Test for Trend, to the encrypted data to produce encrypted results. These are standard algorithms used in genome wide association studies and our proof-of-concept implementation timings are reasonable. We showed that the timings for evaluating the statistical algorithms do not depend on the population size once the correct parameter sizes are fixed and the encrypted genotype or phenotype counts are input. Timings at the smaller parameter size for the various algorithms vary up to roughly 1 second on a standard PC, indicating that these computations are well within reach of being practical for relevant applications and scenarios.

Homomorphic encryption may well be ripe for deployment, to achieve private outsourcing of computation for simple algorithms such as those presented in this paper when applied to modestsize data sets. That will require increased effort and focus on high-performance implementations for a range of architectures. In addition, many other interesting avenues for research remain. There is still much work to be done to make homomorphic encryption more efficient at scale and to expand the functionality. In addition, to solve a wide-range of practical privacy problems which arise with cloud services, it will be important to consider various cryptographic building blocks such as secure multiparty computation and other more interactive solutions and the trade-offs between storage and interaction costs. One should also consider how homomorphic encryption can be combined with building blocks such as verifiable computation. Currently homomorphic encryption does not provide a practical solution for operating on data encrypted under multiple keys, for example in the setting of a public database where multiple patients upload data under different keys. Finally, the practical homomorphic encryption schemes presented here rely on hardness assumptions for a class of new problems such as RLWE. It is crucial to continue to study the hardness of these new assumptions and to attack the systems to accurately assess parameter bounds to assure security.

Acknowledgements. We thank Tancrède Lepoint for suggesting the encoding in Section 4.1.

References

[ADCHT13] Erman Ayday, Emiliano De Cristofaro, Jean-Pierre Hubaux, and Gene Tsudik. The Chills and Thrills of Whole Genome Sequencing. Technical report, 2013. `http://infoscience.epfl.ch/record/186866/files/survey.pdf`.

[ARH13] Erman Ayday, Jean Louis Raisaro, and Jean-Pierre Hubaux. Personal use of the genomic data: Privacy vs. storage cost. In *Proceedings of IEEE Global Communications Conference, Exhibition and Industry Forum (Globecom)*, 2013.

[BAFM12] Marina Blanton, Mikhail Atallah, Keith Frikken, and Qutaibah Malluhi. Secure and efficient outsourcing of sequence comparisons. In *European Symposium on Research in Computer Security (ESORICS)*, pages 505–522. Springer Berlin/Heidelberg, 2012.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[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, *IMA Int. Conf.*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.

[BLN14] Joppe W. Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 2014. to appear, MSR-TR-2013-81.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *FOCS*, pages 97–106. IEEE, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS*, pages 1–12. ACM, 2014.

[dbG] Database of Genotypes and Phenotypes (dbGaP). `http://www.ncbi.nlm.nih.gov/gap/`.

[DCFT13] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. Secure genomic testing with size-and position-hiding private substring matching. In *Proceedings of the 2013 ACM Workshop on Privacy in the Electronic Society (WPES 2013)*. ACM, 2013.

[EBI] European Bioinformatics Institute. `http://www.ebi.ac.uk/`, (accessed 30 October 2013).

[FSU11] Stephen E Fienberg, Aleksandra Slavkovic, and Caroline Uhler. Privacy preserving GWAS data sharing. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 628–635. IEEE, 2011.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[GLN13] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2013.

[Glo13] Creating a global alliance to enable responsible sharing of genomic and clinical data, 2013. White Paper, `http://www.broadinstitute.org/files/news/pdfs/GAWhitePaperJune3.pdf`.

[GMG$^+$13] Melissa Gymrek, Amy L McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, 2013.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.

[HAHT13] Mathias Humbert, Erman Ayday, Jean-Pierre Hubaux, and Amalio Telenti. Addressing the concerns of the lacks family: quantification of kin genomic privacy. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1141–1152. ACM, 2013.

[ICG] International cancer genome consortium (ICGC). `http://www.icgc.org`.

[IRD] International rare diseases research consortium (IRDiRC). `http://www.irdirc.org`.

[Jap] DNA Data Bank Of Japan. `http://www.ddbj.nig.ac.jp/`.

[JS13] Aaron Johnson and Vitaly Shmatikov. Privacy-preserving data exploration in genome-wide association studies. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1079–1087. ACM, 2013.

[LN14a] Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2014.

[LN14b] Adriana López-Alt and Michael Naehrig. Large integer plaintexts in ring-based fully homomorphic encryption. In preparation, 2014.

[LNV11] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM cloud computing security workshop*, pages 113–124. ACM, 2011.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *STOC*, pages 1219–1234. ACM, 2012.

[MCC$^+$11] Catherine A McCarty, Rex L Chisholm, Christopher G Chute, Iftikhar J Kullo, Gail P Jarvik, Eric B Larson, Rongling Li, Daniel R Masys, Marylyn D Ritchie, Dan M Roden, et al. The emerge network: a consortium of biorepositories linked to electronic medical records data for conducting genomic studies. *BMC medical genomics*, 4(1):13, 2011.

[PH08] Mee Young Park and Trevor Hastie. Penalized logistic regression for detecting gene interactions. *Biostatistics*, 9(1):30–50, 2008.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2011.

[TGP] A map of human genome variation from population-scale sequencing. *Nature*, 467:1061–1073. `http://www.1000genomes.org`.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[WLW$^+$09] Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning your identity and disease from research papers: Information leaks in genome wide association study. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 534–544, New York, NY, USA, 2009. ACM.

[YSK^{+}13] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM Cloud computing security workshop*, pages 65–76. ACM, 2013.