University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Secure Remote Genome-wide Association Studies Using Fully Homomorphic Encryption

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Joey Anrdrea M. Cruz

June 2017

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Secure Remote Genome-wide Association Studies Using Fully Homomorphic Encryption" prepared and submitted by Joey Anrdrea M. Cruz in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Richard Bryann L. Chua, M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*cand.*)		
2. Avegail D. Carpio, M.Sc.		
3. Perlita E. Gasmen, M.Sc. (*cand.*)		
4. Marvin John C. Ignacio, M.Sc. (*cand.*)		
5. Ma. Sheila A. Magboo, M.Sc.		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

As genomic data become more widely available, the need for powerful computing resources to process the large volume of data becomes more critical. As a result, researchers are now gearing towards cloud computing. However, there is loss of direct control by the researchers upon their data as they outsource computations, leaving the donors of genomic data vulnerable to exploitation. Homomorphic encryption has become a technique of interest in providing genomic privacy in outsourced computation as it preserves the utility of data in mathematical operations. It is a cryptographic technique that allows the encryption of data and then carrying out computation without decryption. This study proposes a scheme that uses fully homomorphic encryption, using Simple Arithmetic Encrypted Library (SEAL), developed by Microsoft Research.

Keywords: homomorphic encryption, genome-wide association studies, genomic privacy

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

The causal relationship between genetic variations across individuals within a species and the phenotypic differences observed among them is of fundamental biological interest. Since 2005, genome-wide association studies (GWAS) have shown to a significance that susceptibility of individuals to some diseases are associated to certain sequences—i.e. of bases adenine (A), cytosine (C), thymine (T), and guanine (G)—found in their DNA. Many of these studies have revealed correlation to previously unsuspected genes and thus have helped formulate new hypotheses for investigation about disease mechanisms and corresponding treatment targets. [1, 2, 3, 4]

In particular, GWAS investigate the relationship between genetic variations and disease susceptibility through tests of genetic association such as the χ^2 test [1]. Additionally, because of patterns of correlation in DNA sequences, as quantified by linkage disequilibrium, a subset of these many SNPs can be used to show that one or more variants explain part of the genetic risk for a disease [5, 3].

The selection of the samples to be used in the study can affect the accuracy of the results of the study. For this reason, quality control procedures, usually through the computation of and consequent decision-making based on quality control metrics such as minor allele frequency, Hardy-Weinberg equilibrium, and heterozygosity rate, are conducted. [6, 7, 8]

The modern unit of genetic variation is the Single Nucleotide Polymorphism (SNP). SNPs are single base pair changes in the DNA sequence. In GWAS, SNPs are used as markers of a genomic region. They are, by far, the most abundant form of genetic variation in the human genome; the others range from insertions, deletions, and rearrangements. That is, most all the differences between the DNA sequences of any two people occur as SNPs. SNPs can have functional consequences, causing amino acid changes, changes to mRNA transcript stability, and changes to transcription factor binding affinity.

GWAS are feasible because millions of human DNA sequence variations have now been cataloged. New technologies that can assay over one million variants rapidly and accurately have been developed as well. It is due to these high-throughput sequencing technologies that more genetic data are becoming available for study. [1, 2, 3, 4]

Researchers are now gearing towards cloud computing, utilizing a more powerful outsourced computing resource which should accommodate the great volume of data now readily accessible. A very important consideration, however, is the loss of direct control by the researchers upon their data as they outsource computations. In reality, while this should accelerate the processing of data greatly, it leaves these very sensitive genomic data, and consequently its donors, vulnerable to exploitation. [9] Studies have shown that removal of the donors' personal information does not solve the critical problem of privacy, as the genomic data themselves can be used to infer a person's identity. [10, 11]

B. Statement of the Problem

Genome-wide association studies investigate associations between genetic variations across individuals and the phenotypic differences observed among them. These studies involve conducting tests of association which consist of operations on genetic variation data, usually in the form of SNPs.

An important consideration in conducting GWAS, and in genomics as a whole, is that the genome of an individual is highly sensitive data. A malicious entity with knowledge of such can infer an individual's tendency to manifest a genetically associated trait, like his/her disease susceptibility. The entity might then use these inferred information to discriminate against or exploit the individual. Furthermore, given the hereditary nature of genetic material, knowledge of an individual's genome additionally reveals sensitive information about his family, hence similarly exposing them to these kinds of threats.

In this regard, research institutions must handle the important task of keeping their subjects' genomes secure. In the absence of this assurance, the progress of research is impeded by the distrust the institution might receive from prospective subjects, and by the difficulty by which it meets its legal obligations. In other words, the security framework of a genomic research institution is a very important factor that affects the progress of its activities. The same can be said for genomic research as an entire field of study. [12]

Advances in genomic research can largely be attributed to efficient use of computing resources. As genomic data become more widely available, the demand for outsourced, possibly unsafe but more practical, computing power is ever increasing. As a result, the need for genomic data security becomes more critical. [9]

Schemes based on cryptographic techniques can possibly solve the problem of genomic data privacy in GWAS. However, the need to decrypt the data in order for meaningful computation to take place is found in most of these techniques and poses a threat to the privacy of research subjects. In this regard, homomorphic encryption has become a technique of interest as it preserves the utility of data in mathematical operations. It is a cryptographic technique that allows the encryption of data and then carrying out computation without decryption. By this property, it presents a viable option for security in outsourced computation.

C. Objectives of the Study

This research will create a GWAS computing tool that uses homomorphic encryption to perform private computations on genomic data. These computations are to be performed by a server hosted in a cloud environment. The tool will allow the user to do the following:

1. encode and encrypt SNP data, then upload it to the cloud environment

2. perform the following GWAS computations with the uploaded encrypted

SNP data:

(a) χ^2 test statistic

(b) allelic odds ratio

(c) linkage disequilibrium

(d) minor allele frequency

(e) Hardy-Weinberg equilibrium

(f) heterozygosity rate

3. download, decrypt and display the computation results

D. Significance of the Project

In conjunction with policy-based privacy preservation measures, cryptographic solutions, such as that which is detailed in this study, can be used to protect sensitive genomic data and the processing thereof from malicious entities, especially in a research environment that is continuously becoming more borderless due to the rise of cloud-based research.

Specifically, the use of homomorphic encryption can potentially be more relevant and practical in GWAS, as it allows a meaningful encryption of genomic data (i.e. the encrypted form of data are still useful in computation). A large volume of necessarily private arithmetic operations are involved in conducting genome-wide association studies. With the use of such a technique, researchers can securely outsource computation to a possibly unsafe computing resource such as a cloud, all while maximizing its computational capacity. This can be viewed in comparison to other cryptographic techniques that render the data inoperable until decryption.

E. Scope and Limitations

1. The tool will take only Single Nucleotide Polymorphisms (SNPs) as input.

2. The tool will output only the results of the user-selected genomic computation—i.e. intermediate computations and interpretation of results will not be provided.

3. This project will use Simple Encrypted Arithmetic Library (SEAL) developed by Microsoft Research, to perform homomorphic encryption and operations.

F. Assumptions

1. The SNPs are biallelic.

2. The input datasets are in text files.

3. The input datasets follow the format used in the iDASH Privacy & Security Workshop 2015 Secure Genome Analysis Competition.

II. Review of Related Literature

From the time the human genome was first fully sequenced in 2001, genome sequencing costs have significantly declined, from 300 million to a thousand dollars a genome. A completely sequenced genome contains about three billion base pairs. Obtaining data of such a magnitude involves the mapping of a large number of short DNA sequences, called reads, to a reference genome to order these reads correctly. Sequencing thus requires a large amount of resources, including time, storage space, and computational power.

But upon the advance of next-generation sequencing (NGS) technology, available platforms have now become more affordable and efficient, yielding considerably higher throughput. Such an improvement in procuring genomic data has helped the fields of healthcare (e.g. genome-based personalized medicine), research (e.g. genome-wide association studies), consumer services (e.g. ancestry determination), and forensics (e.g. criminal investigations). Most existing literature proposing methods or protocols on protecting genomic information arise as a response to these rapidly advancing genome sequencing technologies, and therefore, wider availability of genomic data.

However, the nature of genomic data opens a security issue for donors (i.e. individuals who have their genome sequenced). An individual is uniquely identified by his/her genome. In fact, it has been shown that even with only a small subset of the genome, 30-80 SNPs out of millions, it is possible to infer the identity of an individual, and estimate his/her disease risk [13]. Consequently, by the nature of genomic data, one might infer information about the individual's family [14]. Thus, the malicious use of these data exposes the individuals, and even their close kin, to possible exploitation by discrimination due to their genomic predisposition.

Numerous works identify approaches that illustrate how such threats to privacy can be realized. Even when the data are anonymized or de-identified (i.e. explicit identifiers are removed), identity information can still be inferred as the data themselves are an individual's identity code. The standard anonymization/deidentification approach to privacy is no longer deemed applicable to genomic data [10]. An approach to re-identification using inferred phenotypes from public data and a list of known associations between genotype and phenotypic traits [11] can be used by malicious entities, and can become more practical as the list of associations grow with further study [15]. Furthermore, the threat of re-identifying anonymized genotype data is compounded by the size of online genotype repositories now available [16]. Aggregated pools of genomic data, including GWAS statistics, can also leak private information. For GWAS, specifically, the presence of a participant in a case group can be inferred using statistical methods from the group's aggregated genomic data [17, 18, 16, 19]. The use of genomic data-sharing beacons also present privacy risks for genomic data donors. These beacons are web servers that answer allele-presence queries—i.e. whether the beacon contains a certain allele in a specified location. It is shown by Shringarpure and Bustamante [20] that through statistical methods, the presence of an individual's genome in a beacon can be inferred with significant power by performing multiple queries. The number of queries necessary depends on the number of individuals in the beacon, given the desired statistical power.

Besides anonymization and other policy-based solutions, other methods aimed at protecting genomic data include cryptography-based approaches, differential privacy, and computation partitioning.

Cassa et al. [21] present a scheme that cryptographically protects and anonymizes an individual's sequence data when it is transmitted between a biorepository and an external sequencing facility using a shared secret key that is derived from a subset of the individual's genetic sequence. This preselected subset of SNPs are naturally known to both parties and consists of highly polymorphic SNPs to ensure high entropy for the generated secret key to be used in the symmetric encryption of the sequenced data. The main advantage of this approach is that it does not require transmission of patient identifiers and passwords to and from sequencing facilities, and a public key infrastructure.

Most cryptographic schemes work under the assumption that the adversary has limited computational resources. When this condition is not true, such schemes are left vulnerable to brute-force attacks—i.e. the adversary permutes across all possible keys. GenoGuard, a scheme based on honey encryption designed by Huang et al. [22], was proposed to especially protect genomic data against brute-force attacks efficiently. A notable property of honey encryption is that when a ciphertext is decrypted with an incorrect key, the result is a plausible-looking plaintext. Additionally, GenoGuard is designed to protect the data against attackers who have side information regarding the target individual.

We also note here that the above works, unlike the proceeding ones, focus less on providing mechanisms for secure computation on genomic data and more on its secure and privacy-preserving outsourced storage.

Ayday et al. [23] and Barman et al. [24] present privacy-preserving disease susceptibility testing using genomic data, a task that involves weighted averaging across SNPs.

Ayday et al. [23] propose a privacy-preserving disease susceptibility test using homomorphic encryption and proxy re-encryption. It details an architecture between a patient, a medical unit, and a storage and processing unit (SPU). Proxy re-encryption is a mechanism through which a patient's secret key can be randomly divided into two shares in such a way that using one of the two shares of the secret key partially decrypts the data, and subsequently using the other share recovers the original message entirely. This mechanism is used to divide the process of retrieving the patient's SNPs into two steps of authorization: one from the SPU and the other from the patient. However, if only computations are necessary, the SPU sends encrypted SNPs to the medical unit who will compute disease susceptibility locally using homomorphic operations and send back the encrypted result to the SPU for decryption.

Barman et al. [24] propose countermeasures for two types of attacks on generic system model architecture between a medical center, a data center, and a patient used in most literature: a test inference attack and an active SNP retrieval attack. A test inference attack can be performed by a data center trying to infer the nature of an ongoing disease susceptibility test for a patient. The disclosure of such information can tell an adversary what disease a patient is being tested for. This attack is mitigated by padding the request for SNPs made by the medical center with dummy SNPs which are not going to be factored in the susceptibility computation. A passive SNP retrieval attack can be performed by a medical center trying to infer a patient's SNPs by manipulating the SNP weights for a test in such a way that will disclose the exact value of a SNP. This can be done by setting all but one SNP weights to zero. The proposed solution prevents this attack by using a protocol which requires the medical center to convince the data center that the test is not a SNP retrieval attack by sending a subset of the SNP weights to be used, and allowing the data center to homomorphically compute upon this subset and send the encryption of the remaining requested SNPs to the medical center. The medical center will then likewise homomorphically compute upon them and combine the two partial results for decryption.

Zhao et al. [25] present a cryptographic approach that supports the search of genomic signatures associated with a certain disease using the computational power of a commercial cloud. To protect the data against attackers, the scheme proposes site-wise encryption—i.e. encoded genome sequences are encrypted before they are uploaded to commercial clouds. It uses the Advanced Encryption Standard (AES), which uses a symmetric key that can be used both for encryption and decryption. To perform the query, the user must also encrypt the query signature using the same key before it is similarly uploaded to the cloud, where simple string matching can be done. To thwart the threat of an attacker who observes queries and their corresponding frequencies, a modification of the scheme that introduces a random bit string in encryption both of the reference and query records is also presented. However, we note that both these approaches are specifically tailored for signature matching and are not useful in other tasks such as

GWAS.

In genomic research, genome-wide association studies (GWAS) investigate associations between genetic variations among individuals and the traits they manifest. The following works present approaches to privacy-preserving GWAS.

Simmons et al. [26], Yu et al. [27], and Tramer et al. [28] use the concept of differential privacy, a form of data perturbation, in genome-wide association studies (GWAS). The main advantage of differential privacy is that formal guarantees of privacy can be made based on mathematics while making minimal assumptions. Intuitively, this kind of privacy guarantees that an analysis performed on any dataset is statistically indistinguishable from the same analysis performed on any dataset that differs in any individual's disease status. It entails selecting a privacy budget, denoted by ϵ, which controls the level of privacy guaranteed to the participants in the study. The closer ϵ is to zero, the more privacy is guaranteed.

Simmons et al. use phenotypic differential privacy as a formal definition of privacy regarding private information about individuals (e.g. disease status). In the selection of SNPs that are highly associated with a disease, the EIGENSTRAT statistic is used. The phenotypically differentially private statistic is estimated for all SNPs, using a technique based on the Laplacian perturbation mechanism, which draws noise from the Laplace distribution. The highest-scoring SNPs are then returned.

Yu et al. compute differentially private genotypic and allelic χ^2 test statistics and similarly returns the results for only a subset of the SNPs. It uses the exponential perturbation mechanism under the motivation that the return of relevant SNPs based merely on their rankings, as the case is in the Laplacian perturbation mechanism, reveals to the attacker that the released SNPs have higher scores than all the other SNPs regardless of the exact values of the computed statistics. The exponential perturbation mechanism, in selecting the SNPs to be released, instead randomly samples across all SNPs whose respective probabilities are weighted based on an exponential function of their scores.

Tramer et al. propose a relaxation of differential privacy to reconcile between utility and privacy in genomic data. In the context of protection against membership disclosure, a re-identification attack where the results of a computation significantly increases an adversary's belief that an entity belongs to a dataset, differential privacy threat models usually make the assumption that for entities about which the adversary has no certain membership knowledge, the adversary has an arbitrary prior belief that the entity belongs to the dataset. The scheme proposed by Tramer et al. relaxes this assumption as it is very unrealistic for an adversary to have high confidence about its belief for all entities a priori. It focuses on the protection of these entities whose priors are unknown—i.e. those whose memberships are highly uncertain,—as guaranteeing privacy for entities for which the adversary has a membership belief with high confidence will require the most perturbation, which consequently degrades data utility. To achieve this, the privacy budget is computed with the relaxed adversarial model in consideration. This approach similarly uses the exponential perturbation mechanism.

For approaches using differential privacy, we note that the large number of queries performed in the process, each with its own privacy budget, yield less accurate results. Similarly, the results in the above works show that, as expected, the differentially private results are more accurate—i.e. have higher utility—as ϵ increases—i.e. as privacy decreases,—as privacy in this context may denote introduction of small noise to the output to achieve indistinguishability.

The interest in using outsourced computational power is evident in GWAS as they involve arithmetic operations over large amounts of data (i.e. many SNPs for many individuals). The proceeding works tackle privacy-preserving outsourcing of GWAS using different approaches.

Zhang et al. [29] and Kamm et al. [10] use secure multi-party computation in their respective works. Both use the mechanism of secret sharing, which ensures that each computing party gets a subset of the data that leaks no information on its own, but is meaningful when the data are pooled back together. Kamm et al.

use (n, t)-threshold linear secret sharing, where each private value is divided into n secret shares to be distributed among n computational parties, such that the combination of t or less shares leak no private information, but the joint use of more than t shares reveals the value exactly. Using such a scheme, linear combinations of secret-shared values can be performed by a computational party on its own, and multiplications of secret-shared values requires communication to occur between all parties. Results from Zhang et al. show a practical execution time that is linear in the number of SNPs and that optimizations to the division operation can prove to be useful as it contributes almost the entire runtime. Kamm et al. find similar challenges in the efficiency of floating point operations, and additionally, in performing comparisons and the maximization of the use of parallel executions. The most significant benefits are regarding the privacy provisions of the approach, which give additional guarantees of security for the data donors even when the research projects discontinue operations, since the data are split among other parties. It also provides a way to conduct analysis that fosters collaboration between independent parties as they do not have to release their data: for example, medical institutions with their patient records, and biobanks with their genotype data. Computational efficiency is the biggest drawback to such an approach, as the nature of secure multi-party computation entails a dependency upon network communication, and additionally, making such communication secure.

Works by Wang et al. [30], Lauter et al. [31], Lu et al. [32], Kim and Lauter [33], and Zhang et al. [34] use homomorphic encryption in GWAS. Fully homomorphic encryption (FHE) enables meaningful computation on data without knowledge of the secret decryption key. In these works, all genomic data in the form of SNPs are encoded and encrypted locally before they are uploaded to the cloud which then conducts computations using homomorphic operations and returns encrypted results.

In the implementation of FHE schemes, some amount of noise is introduced into the ciphertext. These tend to grow during homomorphic operations. At a certain point when the noise becomes too large, the ciphertext can no longer be decrypted even with the correct key. An encryption scheme is said to be somewhat homomorphic if it can evaluate a limited number of operations before the noise grows large enough to cause this kind of error. A leveled homomorphic encryption scheme allows for the selection of parameters to enable correct homomorphic computations. This means that for a specific function to be evaluated, these parameters need to be selected in such a way that the scheme can perform the evaluations correctly. Leveled homomorphic encryption fundamentally does what a fully homomorphic scheme can, and can prove to be practical through the optimization of the parameters.

Wang et al. implemented homomorphic computation of exact logistic regression in a framework called HEALER. Exact logistic regression is a method deemed more robust than standard statistical techniques in computing for p-values with respect to disease association of very rare variants from a limited sample. In doing so, Wang et al. used the BGV scheme for fully homomorphic encryption. Results show that HEALER requires very small storage space for every single encrypted value, in comparison to other approaches using homomorphic encryption, and that in contrast with methods that use the concept of differential privacy, it produces more accurate results. The main limitation of the HEALER framework, however, is that it only considers p-value evaluation.

Lauter et al. implemented the Pearson Goodness-of-Fit test, the D' and r^2 measures of linkage disequilibrium, the Estimation Maximization algorithm for haplotyping, and the Cochran-Armitage Test for Trend using an implementation of the leveled homomorphic encryption scheme proposed by López-Alt and Naehrig [35]. The parameters of the scheme are the degree n, modulus q, and distributions χ_{key} and χ_{err}. As such, the scheme operates in the ring of polynomials with integer coefficients and degree less than n. The integer modulus q is used to reduce the coefficients of the polynomials into the set $\{-\lfloor\frac{q}{2}\rfloor, ..., \lfloor\frac{q}{2}\rfloor\}$ and the distributions are used to generate polynomials with small coefficients. In selecting parameters using such a scheme, particularly n and q, security, correctness, and efficiency are taken into consideration. As long as appropriate parameters are selected, results show that the computations will evaluate correctly. Performance assessment shows that construction and encryption of genotype and phenotype tables run linearly in the size of the data set. On the other hand, execution of the statistical algorithms using homomorphic operations on the encrypted tables is independent of the size of the data set and is instead dependent on the parameters set for the encryption scheme.

Lu et al. supported the evaluation of the D' measure of linkage disequilibrium, Hardy-Weinberg equilibrium, and the χ^2 test statistic also using the BGV scheme in their approach. Worthy of particular note in their work is the packing technique they used to represent vectors of integers, which condenses the sequence of homomorphic additions and multiplications involved in the computation of a scalar product into a single homomorphic multiplication. The use of this packing technique effectively decreased runtime, as shown by a comparison to the work of Lauter et al. [31].

Kim and Lauter presented evaluation of algorithms for secure computation of minor allele frequency, the χ^2 test statistic for GWAS, Hamming distance, and Edit distance between genome sequences across two implementations of homomorphic encryption: the BGV scheme and the YASHE scheme. Both schemes take the parameters n as the polynomial degree, q as the ciphertext coefficient modulus, both used similarly as in the work of Lauter et al., and t as the plaintext coefficient modulus. A trade-off between security and performance was observed. YASHE scheme requires larger parameter values to ensure correctness, as opposed to the BGV scheme, and therefore is slower when evaluating deep circuits such as in computing Hamming and Edit distances. However, for relatively lightweight tasks such as computation of minor allele frequency and χ^2 test statistic, YASHE is more efficient, in terms of runtime, since the BGV scheme uses costly modulus-switching operations even for simple homomorphic operations such as addition, and ciphertext size.

Zhang et al. [34] propose a method that also uses the BGV scheme for fully outsourcing the computation of the χ^2 test statistic for GWAS—that is, it also outsources the execution of division operations, a feature not found in most works that use homomorphic encryption, as currently existing FHE schemes usually do not support homomorphic divisions. They propose two methods for secure outsourced divisions, namely, secure errorless division protocol and secure approximation division protocol. Results show that the approximation protocol provides a good trade-off in terms of complexity and accuracy. Specifically, the errorless protocol requires 10, 20, and 5 times in complexity in key generation, encryption, and execution of computation, respectively, compared to the approximation protocol, and its resulting ciphertext is about eight times as large. Still, however, the work remarks that this trade-off is the main limitation to their proposed solution, as accuracy of results depends on parameters of the encryption scheme, for which larger values lead to decrease in efficiency.

Some libraries that implement homomorphic encryption are now publicly available. HELib [36], developed by IBM Research, is one such library that implements the BGV scheme in C++ atop the GNU Multiple Precision Arithmetic Library (GMP) and the Number Theory Library (NTL). It implements ciphertext packing techniques and single instruction multiple data (SIMD) optimizations. Simple Encrypted Arithmetic Library (SEAL) [37], developed by Microsoft Research, on the other hand, is an implementation of the FV scheme [38] also implemented in C++. It has been made publicly available for bionformatics, genomics, and other research purposes. The library, as part of its optimization mechanism, similarly supports an SIMD mechanism, and in addition, implements a parameter chooser for the leveled homomorphic encryption scheme and data encoders specifically designed to accommodate research data used in bioinformatics and genomics.

III. Theoretical Framework

A. Genome-wide association studies (GWAS)

Genome-wide association studies (GWAS) investigate associations between genetic variations across individuals within a species and the phenotypic differences observed among them. The field has evolved over the last ten years into a powerful tool for investigating the genetic architecture of human disease. GWAS in human genetics aim to identify genetic risk factors for common, complex diseases and rare Mendelian diseases. These studies, which provide insight on the biological underpinnings of disease susceptibility, are useful in developing new prevention and treatment strategies.

One of the early successes of GWAS was the identification of the Complement Factor H gene as a major risk factor for age-related macular degeneration (AMD). Understanding the biological basis of genetic effects, such as that demonstrated in that study, plays an important role in developing new pharmacologic therapies. Accordingly, one of the most successful applications of GWAS has been in the area of pharmacology. Pharmacogenetics has the goal of identifying DNA sequence variations that are associated with drug metabolism and efficacy. This has given rise to the field of personalized medicine that aims to tailor healthcare to individual patients based on their genetic background and other biological features. [1]

In achieving such results, tests of genetic association are conducted and are afterwards replicated to be validated. Tests of genetic association are usually performed separately for each individual genetic marker, usually in the form of Single Nucleotide Polymorphisms (SNPs), which are single-nucleotide substitutions at a genetic locus. Each SNP with major (more abundantly occurring) allele A and a minor (less abundantly occurring) allele B can be represented as a contingency table of counts of disease status by either genotype count (e.g. AA, AB, or BB) or allele count (e.g. A or B). Under the null hypothesis of no association with the disease, the relative allele or genotype frequencies are expected to be the same in case and control groups. A test of association is thus given by a simple χ^2 test for independence of the rows and columns of the contingency table, with the corresponding degrees of freedom, depending on whether allele or genotype counts are used.

The success of a genetic association study depends on genotyping a causal polymorphism, either directly or indirectly. Direct genotyping occurs when an actual causal polymorphism is typed. Indirect genotyping occurs when genetic markers that are highly correlated with the causal polymorphism are typed. Correlation, or non-random association, between alleles at two or more genetic loci is referred to as linkage disequilibrium (LD). [4] The LD structure of the human genome was investigated in the HapMap project, and the outcome was a list of SNPs that captured most of the common genomic variation in a number of human populations. [2]

The selection of the samples to be used in the study is of utmost importance and can affect the accuracy of the results and the statistical power of the method from which they result. Hence, various quality control procedures are employed which can tag individual samples or genetic loci for further examination, or for exclusion altogether from the study. [6, 7, 8]

The discovery of hundreds of thousands of single-nucleotide variants, facilitated by the accelerated sequencing of the human genome, the quantification of the correlation of genetic markers, and the ability to accurately genotype numerous markers in an automated and affordable manner made GWAS a reality. [2] As a result, various research institutions now conduct GWAS and find success in locating genetic variations related to diseases. There are now well over 2000 loci that are significantly and robustly associated with one or more complex traits. The number of loci identified per complex trait varies substantially, from a handful for psychiatric diseases like schizophrenia to a hundred or more for inflammatory bowel disease, including Crohn disease, and ulcerative colitis, and stature. [3, 2]

B. χ^2 test statistic

Dichotomous case/control traits are generally analyzed using contingency table methods. The most ubiquitous form used in GWAS is the χ^2 test. The test examines and measures the deviation from the null hypothesis that there is no association between phenotype and genotype classes. Allelic association tests, such as the χ^2 test, examine the association between one allele of the SNP and the phenotype. [1, 4]

The test uses information from the 2×2 allelic contingency table which contains frequency data from M subjects on the alleles of interest, say A and B. N_A^{case} and N_B^{case} are the counts of A and B alleles, respectively, found in the case group. $N_A^{control}$ and $N_B^{control}$ are the corresponding allele counts for the control group.

	Allele Type		Total
	A	B	
Case	N_A^{case}	N_B^{case}	$N^{case} = N_A^{case} + N_B^{case}$
Control	$N_A^{control}$	$N_B^{control}$	$N^{control} = N_A^{control} + N_B^{control}$
Total	$N_A = N_A^{case} + N_A^{control}$	$N_B = N_B^{case} + N_B^{control}$	$2M$

Table 1: Allelic contingency table

In Table 1, the frequency quantities can be further computed from the genotype data as

$$N_A^{case} = 2N_{AA}^{case} + N_{AB}^{case},$$

$$N_B^{case} = 2N_{BB}^{case} + N_{AB}^{case},$$

$$N_A^{control} = 2N_{AA}^{control} + N_{AB}^{control}, and$$

$$N_B^{control} = 2N_{BB}^{control} + N_{AB}^{control}.$$

The χ^2 test statistic [32] can then be written as

$$\frac{2M(N_B^{case}(N_A^{control}+N_B^{control})-N_B^{control}(N_A^{case}+N_B^{case}))^2}{N^{case}N^{control}N_AN_B}$$

The χ^2 test with one degree of freedom is used to evaluate the significance of the result. Such can be done by finding the p-value associated with the computed statistic using a χ^2 p-value calculator. This p-value is the probability of observing a test statistic equal to or greater than the computed test statistic, under the assumption that the null hypothesis of no association is true. This means that lower p-values indicate that if there is, in fact, no association between the allele and the phenotype, the chance of observing the computed result is extremely small. Results of statistical tests are generally called significant and the null hypothesis is rejected if the p-value falls below a predefined level of siginificance, which is nearly always set to 0.05; in which case, 5% of the time, the null hypothesis is rejected when it is true and we detect a false positive. [1]

C. Allelic odds ratio

The odds ratio is used as a measure of effect size in GWAS. [1] The effect of a locus on the probability of getting the disease can be estimated through the odds ratio associated with the SNP. [39] To illustrate, an odds ratio of one indicates no genetic effect. An odds ratio greater than one indicates that the allele is associated to the trait, and an odds ratio less than one implies that the allele affects the trait negatively.

The allelic odds ratio describes the association between disease and allele by comparing the odds of disease in an individual carrying allele A to the odds of disease in an individual carrying allele B. In other words, the odds ratio is the probability of having the disease given one allele divided by the probability of having the disease given the other allele, as also expressed below. [4]

$$\frac{N_A^{case} N_B^{control}}{N_A^{control} N_B^{case}}$$

D. Minor allele frequency

Minor allele frequency (MAF) refers to the frequency at which the less abundant allele occurs in a population. [1] It can be expressed as

$$\frac{\min(N_A, N_B)}{N_A + N_B}$$

In GWAS, the SNPs selected for investigation are filtered based on minor allele frequency because statistical power is extremely low for rare SNPs—i.e SNPs with very low MAF. The threshold chosen for the MAF depends on the size of the study and the effect sizes expected. SNPs with MAF too low to yield reasonable statistical power (e.g. below 0.01) may be removed from the analysis to lighten computational burden. Association signals seen at these rare SNPs are less robust because they are driven by the genotypes of only a few individuals. [8, 6] In a study by Tabangin et al. [40], common SNPs (with MAF < 0.25) exhibited significantly fewer false positives for association than expected by chance, while rare SNPs exhibited more false positives and variability in its number than common SNPs.

E. Hardy-Weinberg equilibrium

In the absence of migration, mutation, natural selection, and assortative mating, genotype frequencies at any locus are a simple function of allele frequencies. This phenomenon is termed Hardy-Weinberg equilibrium (HWE). [41] Under the assumption of HWE, allele and genotype frequencies can be estimated from one generation to the next. [8]

If A and B are the alleles associated to the locus of interest, let N_{AA}, N_{AB}, N_{BB} be the observed counts for the genotypes AA, AB, and BB, respectively. Furthermore, let

$$p_{AA} = \frac{N_{AA}}{M},\ p_{AB} = \frac{N_{AB}}{M},\ p_{BB} = \frac{N_{BB}}{M}$$

denote the corresponding genotype frequencies. Under the assumption of HWE, the allele frequencies are independent. Hence, the following hold.

$$p_{AA} = p_A^2,\ p_{AB} = 2p_Ap_B,\ p_{BB} = p_B^2$$

Deviations from this assumption is usually measured in GWAS using a χ^2 test, where the expected counts for the genotypes AA, AB, and BB are given by

$$E_{AA} = Mp_A^2,\ E_{AB} = 2Mp_Ap_B,\ E_{BB} = Mp_B^2$$

Deviations of a population from Hardy-Weinberg equilibrium can indicate inbreeding, population stratification, and genotyping errors. Deviation from HWE may be the strongest and most straightforward hint that genotyping may need to be repeated and double-checked. [42] It is not clear how much from these deviations can be attributed to genotyping errors and how much are due to true genotypic frequency deviations from HWE, but examination of plots indicates that most of the extreme deviations are due to poor genotyping. [7]

However, in samples of affected individuals, these deviations can also provide evidence for association. [41, 42] As such, it would be counter-productive to remove these deviating loci from further investigation. It has been consistently noted that SNPs severely out of HWE should therefore not be eliminated from the analysis, but flagged for further analysis after the association analyses are performed. It is also beneficial to examine HWE in controls separately, as controls should more closely follow the assumptions that lead to HWE than cases, since some true associations are expected to be out of HWE. [8]

F. Linkage disequilibrium

Linkage disequilibrium is the nonrandom association of alleles at different loci. It is generated as a consequence of a number of genetic factors and results in the shared ancestry of genetic material. This means that alleles may seem to be inherited together more often rather than expected by chance. [4] GWAS is based on the premise that a marker allele in linkage disequilibrium with the causal variant should likewise show an association with the trait of interest—that is, to say, a subset of SNPs gives information about most of them. [5, 3]

Linkage disequilibrium between alleles at two loci has been defined in many ways, but all definitions depend on the quantity

$$D_{AB} = p_{AB} - p_A p_B$$

where p_{AB} is the frequency of individuals carrying the alleles A and B at two specified loci, and p_A and p_B are the respective frequencies of those alleles observed in the population. [43] One such definition is Lewontin's D' measure of linkage disequilibrium. D' is defined as the ratio of D_{AB} and its largest possible value D_{max} determined by

$$D_{max} = \begin{cases} \min\{p_A p_b, p_a p_B\} & , D > 0 \\ \min\{p_A p_B, p_a p_b\} & , D < 0 \end{cases}$$

A D' value of zero indicates complete linkage equilibrium, which implies statistical independence between the two markers. [1] This condition bears similarities to Hardy-Weinberg equilibrium in implying statistical independence. The essential feature of HWE is that HWE is established in one generation of random mating. Any initial deviation from HWE disappears immediately. LE differs from HWE, however, because it is not established in one generation of random mating but by many.

Closely linked SNPs tend to be in strong linkage disequilibrium with one another. This is also assumed to be true for alleles that increase the risk of complex inherited diseases. This idea, combined with the development of efficient methods for surveying large numbers of SNPs, has led to the many recent GWAS that have detected SNPs that are significantly associated with breast cancer, colorectal cancer, type 2 diabetes and heart disease, among other diseases. [43]

G. Heterozygosity rate

Heterozygosity rate is the proportion of heterozygous genotypes for a given individual. [6, 7]

As a quality control procedure, the distribution of mean heterozygosity, excluding the sex chromosomes, across all individuals should be inspected to identify individuals with an excessive or reduced proportion of heterozygote genotypes. This deviation may be indicative of DNA sample contamination or inbreeding, respectively. [6] Similarly, a study by Pluzhnikov et al. [44] shows that unusual patterns of outlier heterozygosity rates across SNPs can be useful clues about underlying data-quality problems. Due to the increased success rate and accuracy of modern high-throughput genotyping methodologies, exclusion based on heterozygosity lead to only a small proportion of individuals being excluded from further analysis. In doing so, outliers in autosomal heterozygosity can be identified and excluded. [7]

H. Homomorphic encryption (HE)

Homomorphic encryption refers to a class of encryption schemes that allow computation on ciphertexts that will decrypt to the result of computing on the original plaintexts. Formally, it is defined as follows: Let $\mathcal{M}$ denote the set of the plaintexts and $\mathcal{C}$ denote the set of ciphertexts. An encryption scheme is said to be homomorphic if for any given public-private key pair k, s the encryption function

E and the decryption function D satisfies

$$\forall m1, m2 \in \mathcal{M}, \quad D(E(m1) \bigodot_{\mathcal{C}} E(m2)) = m1 \bigodot_{\mathcal{M}} m2$$

for some operators $\bigodot_{\mathcal{M}}$ in $\mathcal{M}$ and $\bigodot_{\mathcal{C}}$ in $\mathcal{C}$. [45]

A scheme is additively homomorphic if we consider an addition operator $\bigodot_{\mathcal{M}}$. A scheme is multiplicatively homomorphic if we consider a multiplication operator. Furthermore, a scheme is said to be partially homomorphic if it is either additively or multiplicatively homomorphic, but not both. It is said to be fully homomorphic if it is both additively and multiplicatively homomorphic. [46]

I. Simple Encrypted Arithmetic Library (SEAL)

Simple Encrypted Arithmetic Library (SEAL) v2.0 [37] by Microsoft Research is a homomorphic encryption solution made publicly available for bioinformatics, genomics, and other research purposes. SEAL is written in C++, but comes with a C# wrapper library called SEALNET. It is an implementation of the Fan-Vercauteren (FV) scheme [38], a leveled homomorphic encryption scheme, and consists of key generation, encryption, decryption, homomorphic addition, and homomorphic multiplication algorithms. The scheme operates in the ring of polynomials with integer coefficients of a degree less than n. Hence, all elements of this ring can be expressed in the form $\sum_{i=0}^{n-1} a_i x^i$ where $a_i \in \mathbb{Z}$.

SEAL implements this polynomial data structure in a class called `BigPoly`, which can represent big polynomials with large coefficients, and additionally abstracts an array of `BigPoly` objects in a class called `BigPolyArray`. In the following discussion, when we say polynomial, we mean by it a polynomial that is a member of the ring discussed above. In the FV scheme, a plaintext is a polynomial, an instance of the `BigPoly` class, and a freshly encrypted ciphertext is an array of two polynomials, represented by an instance of the `BigPolyArray` class. The secret key is a `BigPoly`, and the public key is a `BigPolyArray` of size two.

The implementation also comes with encoder classes that enable the transformation of data like integers and real numbers into plaintext polynomials. A mechanism for decoding the plaintext polynomials into the original data is likewise implemented. The selection of the encoding scheme to be used depends on the type of data to be represented—e.g binary data, integers, real numbers with fractional parts. SEAL also provides a mechanism that packs several pieces of data in a single message, and uses the Single Instruction Multiple Data (SIMD) paradigm to operate on these messages, as implemented in one of its encoding classes, called `PolyCRTBuilder`. The use of such an approach yields reduced message sizes, encoding, encryption, and decryption times, as opposed to encoding the data individually.

Selecting secure parameters for homomorphic encryption is a complicated task, as one must also take into consideration the correctness and efficiency of the computation results. In this regard, SEAL also implements a mechanism for automatic parameter selection in its `ChooserEvaluator` class. The mechanism takes into account the estimated size of the input, referring to bounds in the lengths of plaintext polynomials, the magnitudes of its coefficients, and the homomorphic operations that are to be performed. The selected parameters will then serve as input into the `generate(encryption_parameters)` function of the `KeyGenerator` class which will yield the public-private key pair to be used in initializing the `Encryptor` and `Decryptor` classes, which contain the `encrypt(plaintext)` and `decrypt(ciphertext)` methods, respectively.

To homomorphically perform mathematical operations on encrypted data using SEAL, the `add(encrypted_data, encrypted_data)` and `multiply(encrypted_data, encrypted_data)` functions of the `Evaluator` class can be used. SEAL also allows homomorphic operations between plaintexts and ciphertexts to enable more efficient computation, as already publicly available data—i.e. those that no longer need be made private—do not have to be encrypted anymore to be utilized in computation with ciphertexts. This kind of computation is supported by SEAL's `add_plain(encrypted_data, plain_data)` and `multiply_plain(encrypted_data, plain_data)` functions.

IV. Design and Implementation

A. Use cases and cryptographic architecture

The user will select a desired genomic computation among those supported by the tool, which can be either of the following: χ^2 test statistic, allelic odds ratio, linkage disequilibrium, minor allele frequency, Hardy-Weinberg equilibrium, and heterozygosity rate.

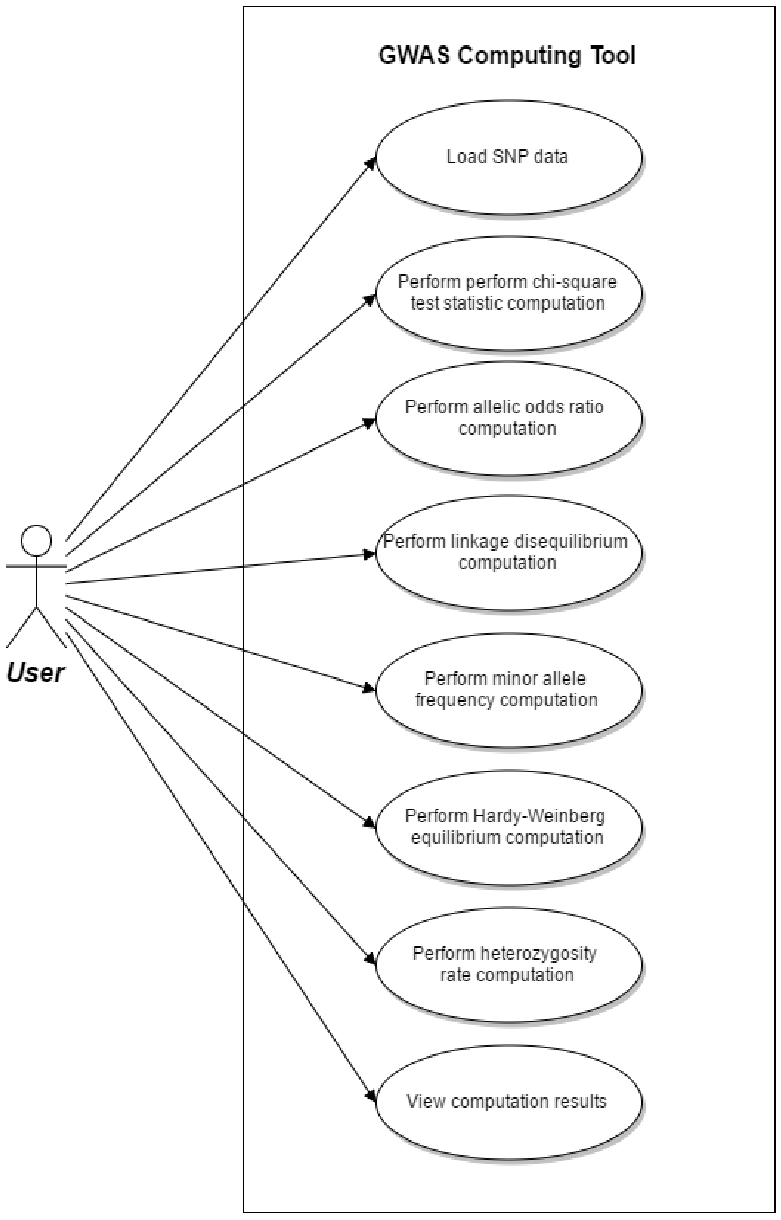

GWAS Computing Tool
Load SNP data
Perform perform chi-square test statistic computation
Perform allelic odds ratio computation
Perform linkage disequilibrium computation
Perform minor allele frequency computation
Perform Hardy-Weinberg equilibrium computation
Perform heterozygosity rate computation
View computation results
User

Figure 1: Use case diagram

The user will then load the SNP data contained in text files for encoding and encryption. For the χ^2 test statistic and allelic odds ratio computations, two input files will be required, one for each the case and the control groups. The rest of the computations will require only one input file.

After the input files have been loaded, the tool will perform the necessary data encoding, encrypting, and uploading into the remote server. The server will then homomorphically operate upon the uploaded ciphertexts and return the results of the computation still in encrypted form. The tool will accordingly download, decrypt, and display the results of the computations. Only the statistics initially specified—that is, no intermediate results—will be displayed to the user.

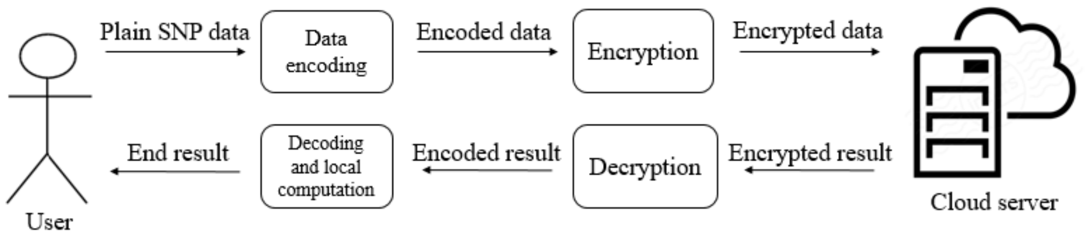
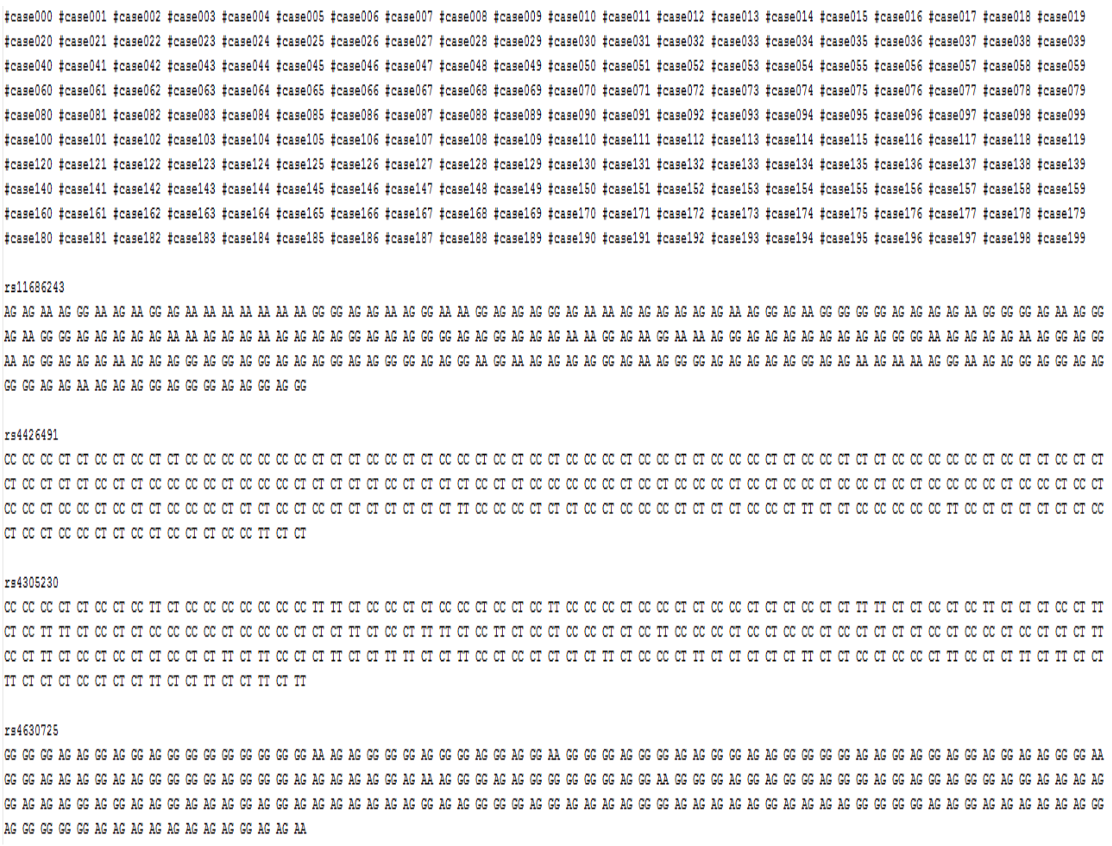


Figure 2: Cryptographic architecture

B. SNP data

The input text files to be accepted by the tool follows the format used in the iDASH Privacy & Security Workshop 2015 Secure Genome Analysis Competition.

The first line in the file contains the identifiers for each of the genotype samples contained in the file. The proceeding lines are all of SNP data, and each SNP is represented by two lines. The first contains the RSID of the SNP, which uniquely identifies the SNP locus. The second enumerates the genotype occurrences for each sample. There are, in this line, as many genotypes as there are identifiers in the first line of the file, and each of the genotypes enumerated correspond to the identifiers in the order they were given.



```
#case000 #case001 #case002 #case003 #case004 #case005 #case006 #case007 #case008 #case009 #case010 #case011 #case012 #case013 #case014 #case015 #case016 #case017 #case018 #case019
#case020 #case021 #case022 #case023 #case024 #case025 #case026 #case027 #case028 #case029 #case030 #case031 #case032 #case033 #case034 #case035 #case036 #case037 #case038 #case039
#case040 #case041 #case042 #case043 #case044 #case045 #case046 #case047 #case048 #case049 #case050 #case051 #case052 #case053 #case054 #case055 #case056 #case057 #case058 #case059
#case060 #case061 #case062 #case063 #case064 #case065 #case066 #case067 #case068 #case069 #case070 #case071 #case072 #case073 #case074 #case075 #case076 #case077 #case078 #case079
#case080 #case081 #case082 #case083 #case084 #case085 #case086 #case087 #case088 #case089 #case090 #case091 #case092 #case093 #case094 #case095 #case096 #case097 #case098 #case099
#case100 #case101 #case102 #case103 #case104 #case105 #case106 #case107 #case108 #case109 #case110 #case111 #case112 #case113 #case114 #case115 #case116 #case117 #case118 #case119
#case120 #case121 #case122 #case123 #case124 #case125 #case126 #case127 #case128 #case129 #case130 #case131 #case132 #case133 #case134 #case135 #case136 #case137 #case138 #case139
#case140 #case141 #case142 #case143 #case144 #case145 #case146 #case147 #case148 #case149 #case150 #case151 #case152 #case153 #case154 #case155 #case156 #case157 #case158 #case159
#case160 #case161 #case162 #case163 #case164 #case165 #case166 #case167 #case168 #case169 #case170 #case171 #case172 #case173 #case174 #case175 #case176 #case177 #case178 #case179
#case180 #case181 #case182 #case183 #case184 #case185 #case186 #case187 #case188 #case189 #case190 #case191 #case192 #case193 #case194 #case195 #case196 #case197 #case198 #case199

rs11686243
AG AG AA AG GG AA AG AA GG AG AA AA AA AA AA AA AA GG GG AG AG AA AG GG AA AA GG AG AG AG GG AG AA AA AG AG AG AG AG AA AG GG AG AA GG GG GG GG AG AG AG AG AA GG GG GG AG AA AG GG
AG AA GG GG AG AG AG AG AG AA AA AG AG AG AA AG AG AG AG GG AG AG AG GG GG AG AG GG AG AG AG AA AA GG AG AA GG AA AA AG GG AG AG AG AG AG AG AG GG GG AA AG AG AG AG AA AG GG AG GG
AA AG GG AG AG AG AA AG AG AG GG AG GG AG GG AG AG AG GG AG AG GG GG AG AG GG AA GG AA AG AG AG GG AG AA AG GG GG AG AG AG AG AG GG AG AG AA AG AA AA AG GG AA AG AG GG AG GG AG AG
GG GG AG AG AA AG AG AG GG AG GG GG AG AG GG AG GG

rs4426491
CC CC CC CT CT CC CT CC CT CT CC CC CC CC CC CC CC CT CT CT CC CC CT CT CC CC CT CC CT CC CT CC CC CC CT CC CC CT CT CC CC CC CT CT CC CC CT CT CT CC CC CC CC CC CT CC CT CT CC CT CT
CT CC CT CT CT CC CT CT CC CC CC CC CT CC CC CC CT CT CT CT CT CC CT CT CT CT CC CT CT CC CC CC CC CC CT CC CT CC CC CC CT CC CT CC CC CT CC CC CT CC CT CC CC CC CC CT CC CC CT CC CT
CC CC CT CC CC CT CC CT CT CC CC CC CT CT CT CC CT CC CT CT CT CT CT CT CT TT CC CC CC CT CT CT CC CT CC CC CC CT CT CT CT CC CC CT TT CT CT CC CC CC CC CC TT CC CT CT CT CT CT CT CC
CT CC CT CC CC CT CT CC CT CC CT CT CC CC TT CT CT

rs4305230
CC CC CC CT CT CC CT CC TT CT CC CC CC CC CC CC CC TT TT CT CC CC CT CT CC CC CT CC CT CC TT CC CC CC CT CC CC CT CT CC CC CT CT CT CC CT CT TT TT CT CT CC CT CC TT CT CT CT CC CT TT
CT CC TT TT CT CC CT CT CC CC CC CC CT CC CC CC CT CT CT TT CT CC CT TT TT CT CC TT CT CC CT CC CC CT CT CC TT CC CC CC CT CC CT CC CC CT CC CT CT CT CT CC CT CC CC CT CC CT CT CT TT
CC CT TT CT CC CT CC CT CT CC CT CT TT CT TT CC CT CT TT CT CT TT TT CT CT TT CC CT CC CT CT CT CT TT CT CC CC CT TT CT CT CT CT CT TT CT CT CC CT CC CC CT TT CC CT CT TT CT TT CT CT
TT CT CT CT CC CT CT CT TT CT CT TT CT CT TT CT TT

rs4630725
GG GG GG AG AG GG AG GG AG GG GG GG GG GG GG GG GG AA AG AG GG GG GG AG GG GG AG GG AG GG AA GG GG GG AG GG GG AG AG GG GG AG AG GG GG GG GG AG AG GG AG GG AG GG AG GG AG AG GG GG AA
GG GG AG AG AG GG AG AG GG GG GG GG AG GG GG GG AG AG AG AG AG GG AG AA AG GG GG AG AG GG GG GG GG GG AG GG AA GG GG GG AG GG AG GG GG AG GG GG AG GG AG GG AG GG GG AG GG AG AG AG AG
GG AG AG AG GG AG GG AG AG GG AG AG AG GG AG GG AG AG AG AG AG AG AG GG AG AG GG GG GG AG GG AG AG AG AG GG GG AG AG AG AG AG GG AG AG AG AG GG GG GG GG AG AG GG AG AG AG AG AG AG GG
AG GG GG GG GG AG AG AG AG AG AG AG AG GG AG AG AA
```

Figure 3: Sample text file containing SNP data in iDASH format

C. Data encoding

When an input text file is loaded into the tool, the SNP data contained in it is encoded into plaintext polynomials, based on the scheme proposed by Lu et al. [32], as follows. Assuming a biallelic locus of, say, alleles A and B, each SNP is represented by four plaintext polynomials, ρ_{fw}^{AA}, ρ_{bw}^{AA}, ρ_{fw}^{AB}, and ρ_{bw}^{AB}. ρ_{fw}^{AA} and ρ_{fw}^{AB} are called forward-packed polynomials; ρ_{bw}^{AA} and ρ_{bw}^{AB} are called backward-packed polynomials. Each of these polynomials are of degree $M-1$, where M is the number of genotype samples given.

The forward-packed polynomials can be expressed as $\rho_{fw}^{X} = \sum_{i=1}^{M} a_i x^{i-1}, X \in \{AA, AB\}$ where a_i is the frequency of the allele A at the ith genotype, if the ith genotype is equal to X. It is zero otherwise. The backward-packed polynomials can be expressed as $\rho_{bw}^{X} = \sum_{i=1}^{M} a_i x^{M-i}, X \in \{AA, AB\}$, where the a_is are defined similarly as above.

To illustrate, take a biallelic locus associated with the alleles A and G. The genotype data AA, AG, AG, AA, GG are encoded as follows:

$$\rho_{fw}^{AA} = 2x^0 + 0x^1 + 0x^2 + 2x^3 + 0x^4, \quad \rho_{bw}^{AA} = 0x^0 + 2x^1 + 0x^2 + 0x^3 + 2x^4,$$
$$\rho_{fw}^{AG} = 0x^0 + 1x^1 + 1x^2 + 0x^3 + 0x^4, \quad \rho_{bw}^{AG} = 0x^0 + 0x^1 + 1x^2 + 1x^3 + 0x^4.$$

The case-control membership statuses of the subjects are also encoded in a polynomial of degree $M - 1$. This polynomial is defined as $\gamma^{case} = \sum_{i=1}^{M} a_i x^{M-i}$, where a_i is one if the ith subject is in the case group and zero if he/she is in the control group. Here, we note that the construction of γ^{case} is similar to that of a backward-packed polynomial. Hence, in the proceeding discussions, we also use it as such. For example, if five subjects have the statuses {case, case, case, control, control}, then the corresponding γ^{case} will be $0x^0 + 0x^1 + 1x^2 + 1x^3 + 1x^4$.

These polynomial encodings of SNP data are then encrypted and uploaded into the remote server where the proceeding computations are performed homomorphically.

D. Scalar product computation

The encoding specified in subsection C. is used to facilitate the computation of a scalar product of two vectors, say $\vec{u}$ and $\vec{v}$. First, we define the notation $\rho_{fw}^{\vec{u}}$ and $\rho_{bw}^{\vec{u}}$. The polynomial $\rho_{fw}^{\vec{u}}$ is $\sum_{i=1}^{n} u_i x^{i-1}$, where n is the length of $\vec{u}$ and u_i is its ith element. Without loss of generality, the polynomial $\rho_{bw}^{\vec{u}}$ is defined as $\sum_{i=1}^{n} u_i x^{n-i}$. We show that the scalar product of two vectors $\vec{u}$ and $\vec{v}$ is given by the coefficient of the middle term of the product $\rho_{fw}^{\vec{u}} \rho_{bw}^{\vec{v}}$. That is, if $\vec{u}$ and $\vec{v}$ are each of length

n, then the middle term is the nth term.

$$\begin{aligned}
\rho_{fw}^{\vec{u}}\rho_{bw}^{\vec{v}} &= (u_1x^0 + u_2x^1 + ... + u_nx^{n-1})(v_nx^0 + v_{n-1}x^1 + ... + v_1x^{n-1}) \\
&= \sum_{k=0}^{n-1}\left(\sum_{i=1}^{k+1} u_iv_{n-k+i-1}\right)x^k + \sum_{k=n}^{2n-2}\left(\sum_{i=k-n+2}^{n} u_iv_{n-k+i-1}\right)x^k \\
&= (u_1v_n)x^0 + (u_1v_{n-1} + u_2v_n)x^1 + ... + \left(\sum_{i=1}^{n} u_iv_i\right)x^{n-1} + ... \\
&\quad + (u_{n-1}v_1 + u_nv_2)x^{2n-1} + (u_nv_1)x^{2n-2}
\end{aligned}$$

In the implementation of the tool, this mechanism is used to arrive at the frequency of an allele/genotype that coincides with particular conditions. To illustrate, the number of homozygous genotypes for the reference allele is the coefficient of the middle term of the polynomial that results from multiplying ρ_{fw}^{AA} with γ^{case}. In the case that no restricting condition is used, the desired frequency can be computed by multiplying the forward packed polynomial to a polynomial of equal degree whose coefficients are all equal to one. Hence, the total number of homozygous genotypes for the reference allele across both the case and control groups is the coefficient of the middle term of the polynomial that results from multiplying ρ_{fw}^{AA} with the polynomial of coefficients equal to one.

E. Single locus allelic contingency table construction

The computation of the χ^2 test statistic, allelic odds ratio, minor allele frequency, and Hardy-Weinberg equilibrium all involve frequency data from an allelic contingency table. Such a contingency table can be constructed with the knowledge of the size of the entire case-control group—i.e. the number of subjects, we denote by M—and the three frequencies N_A^{case}, N_A, and N^{case} derived from the genomic data. Hence, it is sufficient to know these four frequencies from the genomic data in order to compute the mentioned statistics. We note that the quantities M and N^{case} are known to the client, as this is merely the total number of genotype samples given in both the case and control input files and the number of genotype samples in the case input file, respectively.

Let $a_i(\rho)$ be the ith coefficient of the polynomial ρ, and P_n be the polynomial of degree $n-1$ with all coefficients equal to 1. Using the encoding scheme above, the frequencies N_A^{case} and N_A can be computed as follows

$$N_A^{case} = a_M((\rho_{fw}^{AA} + \rho_{fw}^{AB})(\gamma^{case})),$$
$$N_A = a_M((\rho_{fw}^{AA} + \rho_{fw}^{AB})(P_M)),$$

In the above procedure, after the necessary homomorphic multiplications are performed upon the polynomials, they are returned to the client for decryption and construction of the allelic contingency table using the frequencies given by the middle coefficients of the decrypted polynomials. The remaining cells in the table are given by the following:

$$N_B^{case} = 2N^{case} - N_A^{case}$$
$$N_A^{control} = N_A - N_A^{case}$$
$$N_B = 2M - N_A$$
$$N^{control} = M - N^{case}$$
$$N_B^{control} = N_B - N_B^{case} = N^{control} - N_A^{control}$$

F. Computation of statistics

After the construction of the allelic contingency table, the χ^2 test statistic, allelic odds ratio, and minor allele frequency can be computed directly using their respective formulas, as given in Table 2. Hardy-Weinberg equilibrium computation additionally requires the genotype frequencies N_{AA}, N_{AB}, and N_{BB}. N_{AA} and

N_{AB} can be computed remotely and similarly returned to the client as

$$2N_{AA} = a_M((\rho_{fw}^{AA})(P_M))$$
$$N_{AB} = a_M((\rho_{fw}^{AB})(P_M)).$$

N_{BB} is then given by $M - N_{AA} - N_{AB}$.

Statistic	Formula
χ^2 test statistic	$\frac{2M(N_B^{case}(N_A^{control}+N_B^{control})-N_B^{control}(N_A^{case}+N_B^{case}))^2}{N^{case}N^{control}N_A N_B}$
Allelic odds ratio	$\frac{N_A^{case}N_B^{control}}{N_A^{control}N_B^{case}}$
Minor allele frequency	$\frac{\min(N_A,N_B)}{N_A+N_B}$
Hardy-Weinberg equilibrium	$\sum_{X\in\{AA,AB,BB\}} \frac{(N_X-E_X)^2}{E_X}$

Table 2: Formulas for allelic case-control contingency table-based statistics

For two biallelic loci, the first with reference allele A and alternate allele a, and the second with reference allele B and alternate allele b, linkage disequilibrium is defined as

$$D' = \frac{D}{D_{max}}, where$$
$$D = p_{AB} - p_A p_B, and$$
$$D_{max} = \begin{cases} \min\{p_A(1-p_B), (1-p_A)p_B\} & , D > 0 \\ \min\{p_A p_B, (1-p_A)(1-p_B)\} & , D < 0 \end{cases}.$$

The quantities necessary to complete this computation are the proportions p_{AB}, p_A, and p_B. These can be computed as

$$p_{AB} = \frac{2N_{AABB} + N_{AaBB} + N_{AABb}}{2M}$$
$$p_A = \frac{2N_{AA} + N_{Aa} - N_{AaBb}}{2M}$$
$$p_B = \frac{2N_{BB} + N_{Bb} - N_{AaBb}}{2M}$$

.

These required computations can be further broken down in terms of the quantities N_{AABB}, N_{AaBB}, N_{AABb}, N_{AaBb}, $2N_{AA} + N_{Aa}$, and $2N_{BB} + N_{Bb}$. These, in turn, can be computed homomorphically as

$$\begin{aligned}
4N_{AABB} &= a_M((\rho_{fw}^{AA})(\rho_{bw}^{BB})) \\
2N_{AaBB} &= a_M((\rho_{fw}^{Aa})(\rho_{bw}^{BB})) \\
2N_{AABb} &= a_M((\rho_{fw}^{AA})(\rho_{bw}^{Bb})) \\
N_{AaBb} &= a_M((\rho_{fw}^{Aa})(\rho_{bw}^{Bb})) \\
2N_{AA} + N_{Aa} &= a_M((\rho_{fw}^{AA} + \rho_{fw}^{Aa})(P_M)) \\
2N_{BB} + N_{Bb} &= a_M((\rho_{fw}^{BB} + \rho_{fw}^{Bb})(P_M))
\end{aligned}$$

The tool developed in this study outputs the linkage disequililbrium of the loci pairs consisting of the first SNP in the input file and each of the proceeding SNPs given.

The total heterozygosity across N SNPs for several individuals can be computed homomorphically as

$$\rho_{HR} = \sum_{i=1}^{N} \rho_{fw}^{Aa}$$

The heterozygosity rate of the ith individual is given by

$$\frac{a_i(\rho_{HR})}{N}$$

.

G. System Architecture

The two crucial modules used by the tool are the server, which is responsible for performing computations on encrypted genomic data, and the client application, responsible for user interaction, encoding, encrypting, and decrypting data.

All communication between the client and the server is implemented using

Windows Sockets. The user interface of the client application was developed using Windows Forms in Visual C++, under the .NET Framework. All encryption functionalities used were implemented in SEAL.

H. Technical Architecture

The server runs on a Windows 10 machine with the following specifications:

1. Processor: Intel Core i7-4510u @ 2.60GHz

2. Memory: 8GB

3. Operating System: 64-bit

The client runs on a Windows 10 machine with the following specifications:

1. Processor: Intel Core i7-6500U @ 2.50GHz

2. Memory: 8GB

3. Operating System: 64-bit

V. Results

The tool initially displays a choice of six statistics: χ^2 test statistic, allelic odds ratio, linkage disequilibrium, minor allele frequency, Hardy-Weinberg equilibrium, and heterozygosity rate.

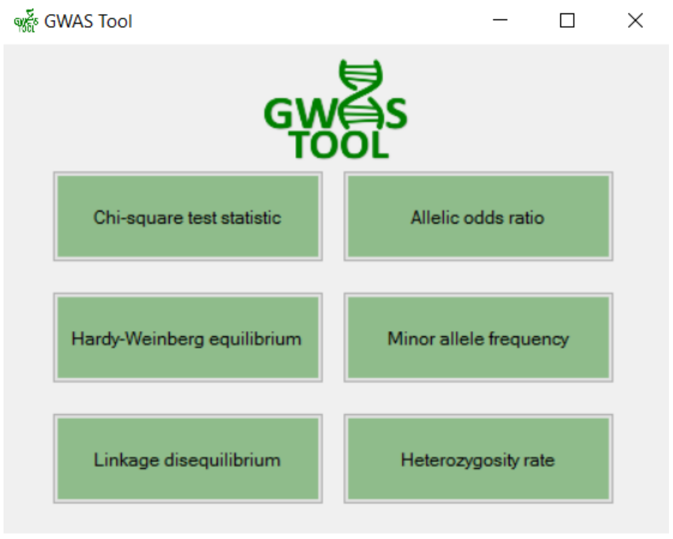


Figure 4: Home Screen

For the choice of χ^2 test statistic and allelic odds ratio, the user is prompted to pick exactly two text files.

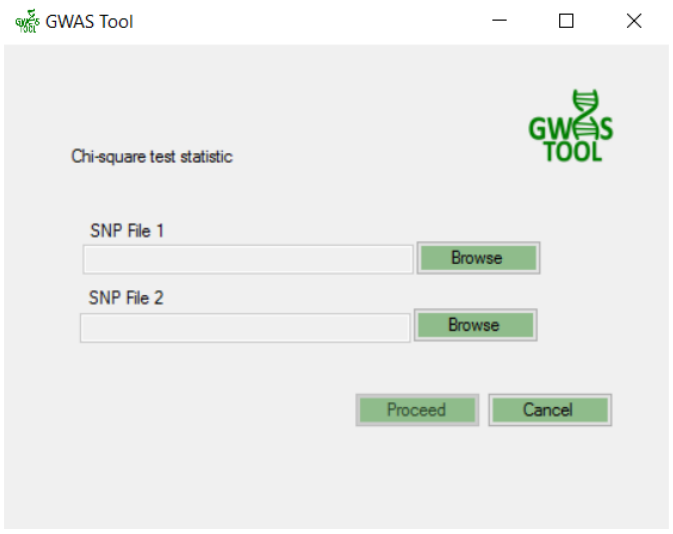


Figure 5: Picking two files

For linkage disequilibrium, minor allele frequency, Hardy-Weinberg equilibrium, and heterozygosity rate, only a single file is required from the user.

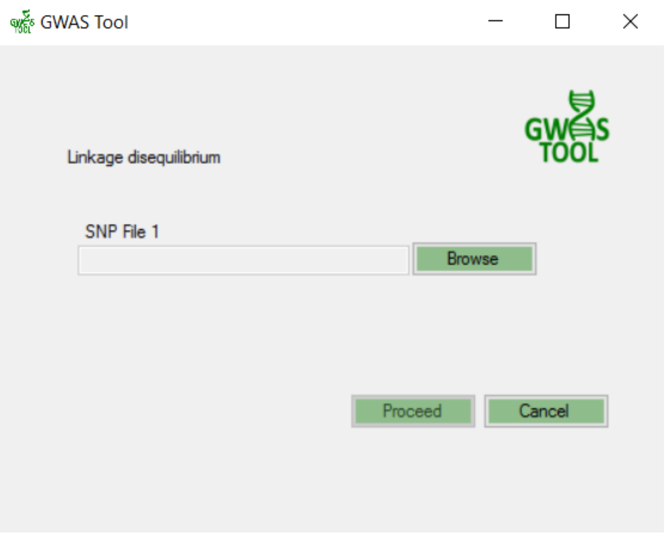
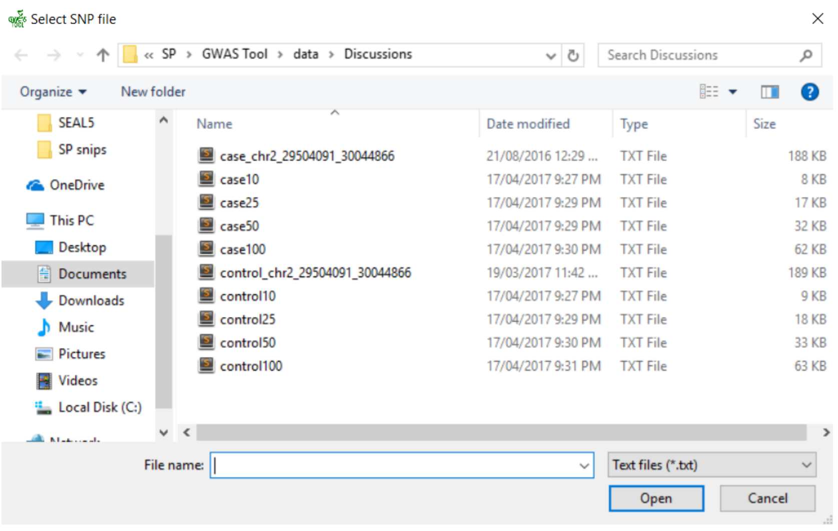


Figure 6: Picking one file

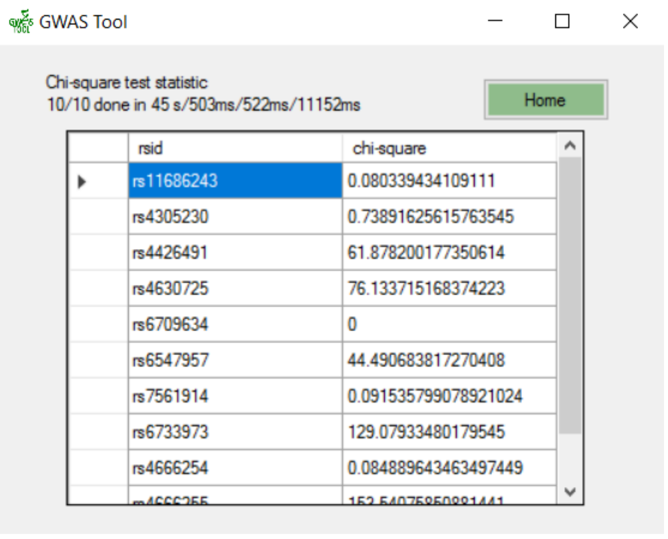
For either case, when the user clicks on the *Browse* button, a file picker dialog will appear.



Name	Date modified	Type	Size
case_chr2_29504091_30044866	21/08/2016 12:29 ...	TXT File	188 KB
case10	17/04/2017 9:27 PM	TXT File	8 KB
case25	17/04/2017 9:29 PM	TXT File	17 KB
case50	17/04/2017 9:29 PM	TXT File	32 KB
case100	17/04/2017 9:30 PM	TXT File	62 KB
control_chr2_29504091_30044866	19/03/2017 11:42 ...	TXT File	189 KB
control10	17/04/2017 9:27 PM	TXT File	9 KB
control25	17/04/2017 9:29 PM	TXT File	18 KB
control50	17/04/2017 9:30 PM	TXT File	33 KB
control100	17/04/2017 9:31 PM	TXT File	63 KB

Figure 7: File picker dialog

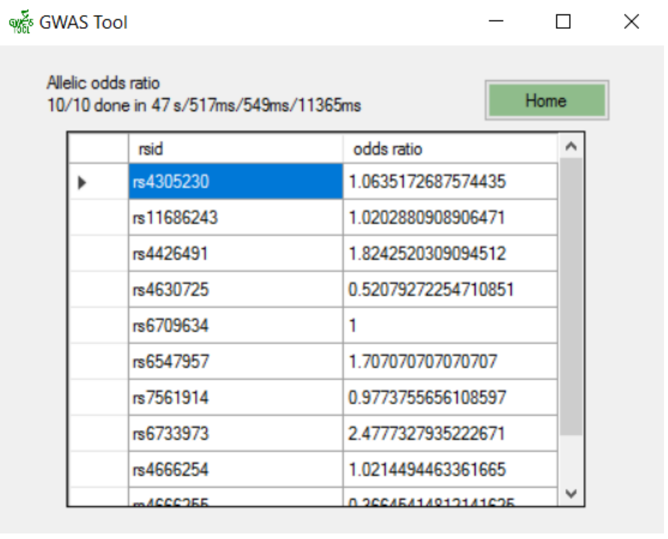
Clicking on the *Cancel* button directs the user back to the home screen. Clicking on the *Proceed* button will commence the computation, i.e. encoding, encryption, uploading, decrypting, and display of results. Figures 8, 9, 10, 11, 12, and 13 show the respective results screens for each of the statistics which contain a table of two columns, one for the SNP identifier and another for the statistic computed. On each of these screens, clicking the *Home* button redirects the user to the home screen. Additionally, four running time metrics are provided in the following format: time elapsed/average encryption time/average decryption time/average remote time



Chi-square test statistic
10/10 done in 45 s/503ms/522ms/11152ms

rsid	chi-square
rs11686243	0.080339434109111
rs4305230	0.73891625615763545
rs4426491	61.878200177350614
rs4630725	76.133715168374223
rs6709634	0
rs6547957	44.490683817270408
rs7561914	0.091535799078921024
rs6733973	129.07933480179545
rs4666254	0.084889643463497449
rs4666255	[illegible]

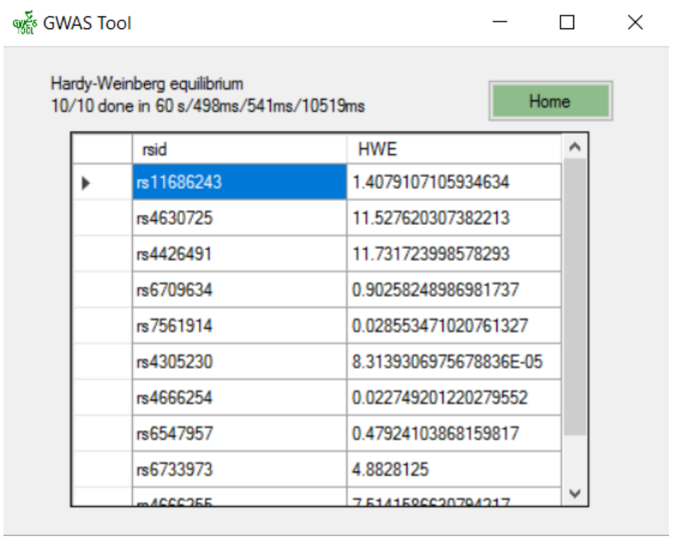
Figure 8: χ^2 test statistic results



Allelic odds ratio
10/10 done in 47 s/517ms/549ms/11365ms

rsid	odds ratio
rs4305230	1.0635172687574435
rs11686243	1.0202880908906471
rs4426491	1.8242520309094512
rs4630725	0.52079272254710851
rs6709634	1
rs6547957	1.707070707070707
rs7561914	0.9773755656108597
rs6733973	2.4777327935222671
rs4666254	1.0214494463361665
rs4666255	[illegible]

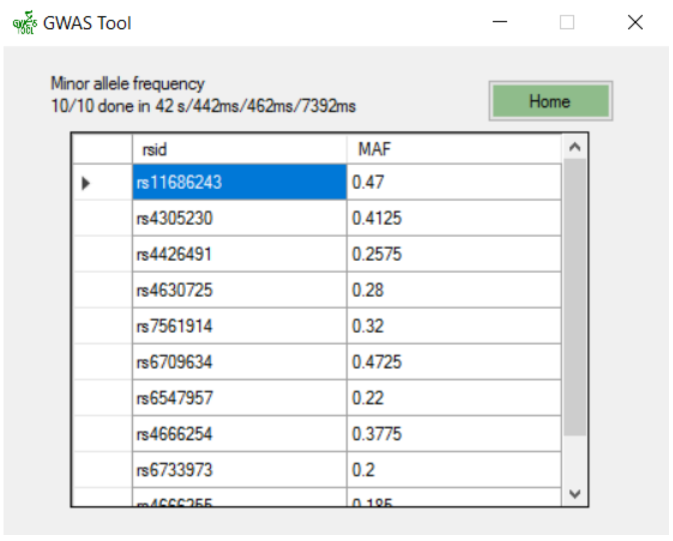
Figure 9: AOR results



Hardy-Weinberg equilibrium
10/10 done in 60 s/498ms/541ms/10519ms

rsid	HWE
rs11686243	1.4079107105934634
rs4630725	11.527620307382213
rs4426491	11.731723998578293
rs6709634	0.90258248986981737
rs7561914	0.028553471020761327
rs4305230	8.3139306975678836E-05
rs4666254	0.022749201220279552
rs6547957	0.47924103868159817
rs6733973	4.8828125
rs4666255	7.5141586620794217

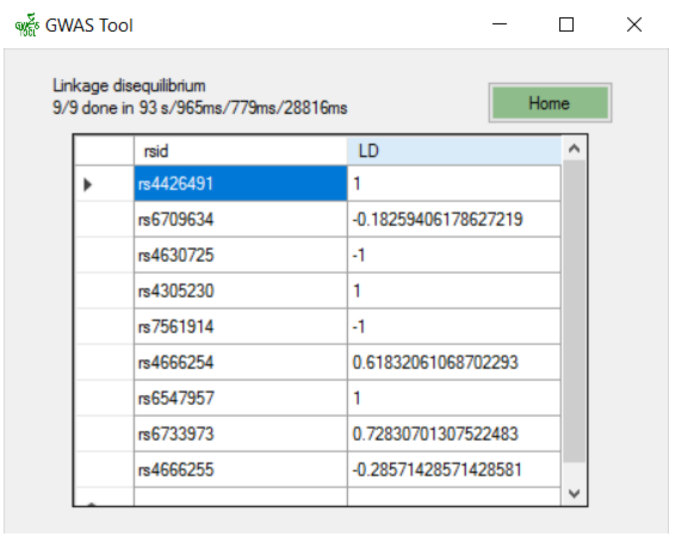
Figure 10: Hardy-Weinberg equilibrium results



Minor allele frequency
10/10 done in 42 s/442ms/462ms/7392ms

rsid	MAF
rs11686243	0.47
rs4305230	0.4125
rs4426491	0.2575
rs4630725	0.28
rs7561914	0.32
rs6709634	0.4725
rs6547957	0.22
rs4666254	0.3775
rs6733973	0.2
rs4666255	0.185

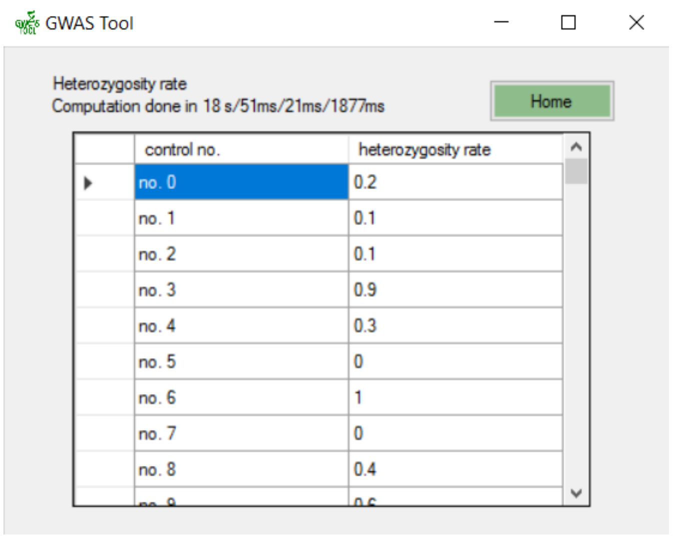
Figure 11: Minor allele frequency results


GWAS Tool

Linkage disequilibrium
9/9 done in 93 s/965ms/779ms/28816ms

rsid	LD
rs4426491	1
rs6709634	-0.182594061786272 19
rs4630725	-1
rs4305230	1
rs7561914	-1
rs4666254	0.61832061068702293
rs6547957	1
rs6733973	0.72830701307522483
rs4666255	-0.28571428571428581

Figure 12: Linkage disequilibrium results


GWAS Tool

Heterozygosity rate
Computation done in 18 s/51ms/21ms/1877ms

control no.	heterozygosity rate
no. 0	0.2
no. 1	0.1
no. 2	0.1
no. 3	0.9
no. 4	0.3
no. 5	0
no. 6	1
no. 7	0
no. 8	0.4

Figure 13: Heterozygosity rate results

VI. Discussions

A. SNP class

To encapsulate the SNP data read from input files together with related encoding methods, the `Snp` class was implemented.

```
#include "stdafx.h"
#include "Snp.h"

using namespace std;

Snp::Snp() {
  this->homo = vector<int>();
  this->hetero = vector<int>();
  this->phenotypes = vector<int>();
}

Snp::Snp(string rsid, string genotypes, vector<int> phenotypes) {
  vector<char> alleles;
  if (genotypes.find(static_cast<char>(Snp::allele::A)) != string::npos) alleles.
    push_back(static_cast<char>(Snp::allele::A));
  if (genotypes.find(static_cast<char>(Snp::allele::C)) != string::npos) alleles.
    push_back(static_cast<char>(Snp::allele::C));
  if (genotypes.find(static_cast<char>(Snp::allele::T)) != string::npos) alleles.
    push_back(static_cast<char>(Snp::allele::T));
  if (genotypes.find(static_cast<char>(Snp::allele::G)) != string::npos) alleles.
    push_back(static_cast<char>(Snp::allele::G));

  sort(alleles.begin(), alleles.end());
  string homo = string(2, alleles[0]);
  string hetero1 = string(1, alleles[0]);
  hetero1 += alleles[1];
  string hetero2 = string(1, alleles[1]);
  hetero2 += alleles[0];

  stringstream ss = stringstream(genotypes);
  string basepair;
  while (getline(ss, basepair, ' ')) {
    if (basepair == homo) {
      this->homo.push_back(2);
      this->hetero.push_back(0);
    }
    else if (basepair == hetero1 || basepair == hetero2) {
      this->homo.push_back(0);
      this->hetero.push_back(1);
    }
    else {
      this->homo.push_back(0);
      this->hetero.push_back(0);
    }
  }

  this->rsid = rsid;
  this->allele1 = (Snp::allele)alleles[0];
  this->allele2 = (Snp::allele)alleles[1];
  this->phenotypes = phenotypes;
}

Snp::~Snp() {}

string Snp::getRsid() {
  return this->rsid;
}

int Snp::getSize() {
  return (int)this->phenotypes.size();
}

Snp::allele Snp::getAllele1() {
  return this->allele1;
}

Snp::allele Snp::getAllele2() {
  return this->allele2;
}

string Snp::getHomoPolyFw() {
  stringstream ss;
  int degree = (int)this->homo.size() - 1;
  for (size_t ctr = 0; ctr <= degree; ctr++) {
    if (ctr > 0) ss << " ";
    ss << homo[degree - ctr] << "x^" << degree - ctr;
    if (ctr < degree) ss << " +";
  }
  return ss.str();
}
```

```cpp
string Snp::getHomoPolyBw() {
  stringstream ss;
  int degree = static_cast<int>(this->homo.size()) - 1;
  for (int ctr = 0; ctr <= degree; ctr++) {
    if (ctr > 0) ss << " ";
    ss << homo[ctr] << "x^" << degree - ctr;
    if (ctr < degree) ss << " +";
  }
  return ss.str();
}

string Snp::getHeteroPolyFw() {
  stringstream ss;
  int degree = static_cast<int>(this->hetero.size()) - 1;
  for (int ctr = 0; ctr <= degree; ctr++) {
    if (ctr > 0) ss << " ";
    ss << hetero[degree - ctr] << "x^" << degree - ctr;
    if (ctr < degree) ss << " +";
  }
  return ss.str();
}

string Snp::getHeteroPolyBw() {
  stringstream ss;
  int degree = static_cast<int>(this->hetero.size()) - 1;
  for (int ctr = 0; ctr <= degree; ctr++) {
    if (ctr > 0) ss << " ";
    ss << hetero[ctr] << "x^" << degree - ctr;
    if (ctr < degree) ss << " +";
  }
  return ss.str();
}

string Snp::getPhenotypePolyBw() {
  stringstream ss;
  int degree = static_cast<int>(this->phenotypes.size()) - 1;
  for (int ctr = 0; ctr <= degree; ctr++) {
    if (ctr > 0) ss << " ";
    ss << phenotypes[ctr] << "x^" << degree - ctr;
    if (ctr < degree) ss << " +";
  }
  return ss.str();
}
```

```
void Snp::add(Snp snp) {
  if (this->rsid == snp.getRsid() && this->allele1 == snp.getAllele1() && this->
    allele2 == snp.getAllele2()) {
    this->homo.insert(this->homo.end(), snp.homo.begin(), snp.homo.end());
    this->hetero.insert(this->hetero.end(), snp.hetero.begin(), snp.hetero.end())
    ;
    this->phenotypes.insert(this->phenotypes.end(), snp.phenotypes.begin(), snp.
    phenotypes.end());
  }
}
```

After construction of an instance of the class, calls to the member functions `getHomoPolyFw()`, `getHomoPolyBw()`, `getHeteroPolyFw()`, `getHeteroPolyBw()`, `getPhenotypePolyFw()`, and `getPhenotypePolyBw()` will generate the forward and backward packed polynomials described in the previous chapter that correspond to the SNP represented by the instance.

B. Encryption

The polynomials constructed above will then be encrypted using SEAL. Selection of encryption parameters are based on the size of the input and the type of computation to be done. In SEAL, these parameters are the polynomial modulus `poly_mod`, the coefficient modulus `coeff_mod`, and the plaintext modulus `plain_mod`. The `coeff_mod` can be appropriately selected by one of SEAL's methods based on the `poly_mod`. The tool is then left to select the `poly_mod` and `plain_mod`.

In Table 3 we show the `poly_mod` and `plain_mod` selected for the computation of statistics based on a single locus contingency table (i.e. χ^2 test statistic, allelic odds ratio, Hardy-Weinberg equilibrium, and minor allele frequency), linkage disequilibrium, and heterozygosity rate.

Statistic type	`poly_mod`	`plain_mod`
Single locus allelic contingency	$1x^{2^{(\log_2 M+2)}}+1$	$2^{(\log_2 M+2)}$
Linkage disequilibrium	$1x^{2^{(\log_2 M+3)}}+1$	$2^{(\log_2 M+4)}$
Heterozygosity rate	$1x^{2^{(\log_2 M+2)}}+1$	$2^{(\log_2 M+1)}$

Table 3: Encryption parameters

In general, after these parameters are selected, key generation and encryption can be done using the following code.

```
//Key generation
KeyGenerator generator(parms);
generator.generate();
BigPolyArray public_key = generator.public_key();
BigPoly secret_key = generator.secret_key();

//Encrypting values
Encryptor encryptor(parms, public_key);
encryptor.encrypt(snp.getHomoPolyFw());
encryptor.encrypt(snp.getHeteroPolyFw());
encryptor.encrypt(snp.getHomoPolyBw());
encryptor.encrypt(snp.getHeteroPolyBw());
```

C. Remote computation

After required ciphertexts are created, they are sent to the server through Windows Sockets. The server performs homomorphic computation upon these using its `Evaluator` class, which uses the same set of parameters detailed in Table 3.

In the following discussion, we use the following variables to denote certain data described in Table 4.

Variable	Description
`homoPolyFw`	encrypted forward packed polynomial of the homozygous genotype
`homoPolyBw`	encrypted backward packed polynomial of the homozygous genotype
`heteroPolyFw`	encrypted forward packed polynomial of the heterozygous genotype
`heteroPolyBw`	encrypted backward packed polynomial of the heterozygous genotype
`onePoly`	the plaintext polynomial P_M

Table 4: Variables and descriptions

The procedures detailed below involve the homomorphic operations required in the computation of each of the statistics and are all performed by the remote server.

C..1 Single locus allelic contingency table

For statistics based on a single locus contingency table, i.e. χ^2 test statistic, allelic odds ratio, minor allele frequency, and Hardy-Weinberg equilibrium a similar sequence of homomorphic operations is followed. Specifically, for the first three statistics, we have the following code.

```
BigPolyArray encryptedsum = evaluator.add(homoPolyFw, heteroPolyFw);
BigPolyArray encryptedacase = evaluator.multiply(encryptedsum, phenotypeBw);
BigPolyArray encrypteda = evaluator.multiply_plain(encryptedsum, onePoly);
```

The ciphertexts `encryptedacase` and `encrypteda` are returned to the client for the corresponding computation of the requested statistic using the formulas given in the previous chapter. These ciphertexts correspond to the polynomials whose middle coefficients are N_A^{case} and N_A, respectively.

For Hardy-Weinberg equililbrium, two additional ciphertexts are sent back to the client, `encryptedaa` and `encryptedab`, computed as follows. These likewise correspond to the polynomial whose middle coefficients are the frequencies N_{AA} and N_{AB}.

```
BigPolyArray encryptedaa = evaluator.multiply_plain(homoPolyFw, onePoly);
BigPolyArray encryptedab = evaluator.multiply_plain(heterPolyFw, onePoly);
```

C..2 Linkage disequilibrium

In the remote computation of linkage disequilibrium, we have the following code. In the notation used, the number indicated in the variable name refers to a unique locus.

```
BigPolyArray homohomo = evaluator.multiply(homoPolyFw1, homoPolyBw2);
BigPolyArray heterohomo = evaluator.multiply(heteroPolyFw1, homoPolyBw2);
BigPolyArray homohetero = evaluator.multiply(homoPolyFw1, heteroPolyBw2);
BigPolyArray heterohetero evaluator.multiply(heteroPolyFw1, heteroPolyBw2);

BigPolyArray encryptedsum1 = evaluator.add(homoPolyFw1, heteroPolyFw1);
BigPolyArray a = evaluator.multiply_plain(encryptedsum1, onePoly);


BigPolyArray encryptedsum2 = evaluator.add(homoPolyFw2, heteroPolyFw2);
BigPolyArray b evaluator.multiply_plain(encryptedsum2, onePoly);
```

The ciphertexts `homohomo`, `heterohomo`, `homohetero`, `heterohetero`, `a`, and `b` are returned to the client. These are the encryptions of the polynomials whose middle coefficients are $4N_{AABB}$, $2N_{AaBB}$, $2N_{AABb}$, N_{AaBb}, $2N_{AA}+N_{Aa}$, and $2N_{BB}+N_{Bb}$, the necessary frequencies in the computation of linkage disequilibrium.

C..3 Heterozygosity rate

In the computation of heterozygosity rate, only a sequence of addition on forward packed polynomials are necessary.

```
BigPolyArray sum = ciphertexts[0];
ciphertexts.erase(ciphertexts.begin());

for (BigPolyArray ciphertext : ciphertexts) {
  sum = evaluator.add(sum, ciphertext);
}
```

The ciphertext `sum` is returned to the client. The ith coefficient of the decryption of this ciphertext is the number of heterozygous genotypes of the ith individual. To output the heterozygosity rate, this quantity is divided by the total number of SNPs included in the computation.

D. Decryption

When the client application receives all ciphertexts expected from the requested computation, it decrypts, does a few arithmetic operations based on the sufficient statistics, and displays only the end result on the user interface. The following code is used to decrypt ciphertexts.

```
Decryptor decryptor(parms, secret_key);
BigPoly decrypted = decryptor.decrypt(ciphertext);
```

To access the coefficients of the decrypted polynomial, the syntax used is similar to that of accessing the element an array. For example, when the frequency of interest is the middle—i.e. the Mth—coefficient of the polynomial `decrypted`, it is accessed using the following code.

```
decrypted[this->snp.getSize()].to_double());
```

E. GUI and multithreading

Both the user interface and multithreading feature were implemented in the .NET Framework. Because the native C++ threading library `<thread>` is not supported in .NET, particular effort was made into interfacing between the native C++ classes that implement the homomorphic computations and the managed classes (i.e. those constructed under .NET) for the user interface.

Specifically, a new managed class, `SnpString`, was introduced to represent a SNP only using objects in C++ .NET that can be accepted as parameters to a .NET Thread. When an input file is read, the SNPs contained in it are first represented as a `SnpString`. Then, the contents of each `SnpString` is used as a parameter to a `ParametrizedThreadStart`. In that thread, a `Snp` is constructed only then to be homomorphically encrypted and operated upon, as SEAL uses objects in native C++. As a result, whenever applicable—that is, for the χ^2 test statistic, allelic odds ratio, Hardy-Weinberg equilibrium, minor allele frequency, and linkage disequilibrium,—computations can occur in parallel. For a given value `MAX_THREAD`, the tool can execute `MAX_THREAD` threads in parallel. The following code shows how multithreading is achieved in the computation of the χ^2 test statistic.

```
List<Thread^>^ threads = gcnew List<Thread^>();

for (int ctr = 0; ctr < cc.size() && executing; ctr++) {
  vector<SnpString> ccpair = cc[ctr];
  List<String^>^ rsids = gcnew List<String^>();
  rsids->Add(gcnew String(ccpair[0].getRsid().c_str()));
  rsids->Add(gcnew String(ccpair[1].getRsid().c_str()));

  List<String^>^ genos = gcnew List<String^>();
  genos->Add(gcnew String(ccpair[0].getGenotypes().c_str()));
  genos->Add(gcnew String(ccpair[1].getGenotypes().c_str()));

  List<int>^ sizes = gcnew List<int>();
  sizes->Add(ccpair[0].getSize());
  sizes->Add(ccpair[1].getSize());

  Tuple<List<String^>^, List<String^>^, List<int>^>^ input = gcnew Tuple<List<
    String^>^, List<String^>^, List<int>^>(rsids, genos, sizes);
  ParameterizedThreadStart^ myThreadDelegate = gcnew ParameterizedThreadStart(
    this, &GWASGUI::chisq);
  Thread^ thread1 = gcnew Thread(myThreadDelegate);

  thread1->Start(input);
  threads->Add(thread1);

  if (threads->Count == MAX_THREAD) {
    threads[0]->Join();
    threads->RemoveAt(0);
  }
}

for (int ctr2 = 0; ctr2 < threads->Count; ctr2++) {
  threads[ctr2]->Join();
}
```

In the above code, each call to `GWASGUI::chisq` in a thread corresponds to a parallel execution of the single locus allelic contingency table procedure described above. A similar mechanism does the same for allelic odds ratio, Hardy-Weinberg equilibrium, minor allele frequency, and linkage disequilibrium.

F. Timing

In this subsection, we discuss the execution time of the homomorphic computations. Table 5 details the metrics used in the discussion.

Metric	Description
TTE	Total time elapsed from the beginning of the computation to its completion
AET	Average encryption time
ADT	Average decryption time
ART	Average remote time (i.e. time elapsed waiting for the server)

Table 5: Timing metrics

In the computation of the χ^2 test statistic, allelic odds ratio, Hardy-Weinberg equilibrium, minor allele frequency, and linkage disequilibrium, an input corresponds to a sequence of identical transactions to the server. That is, for each SNP in the file, the same computation occurs and these computations are independent of each other. Because homomorphic computations can be costly with respect to execution time, we explore the option of parallel computation, the implementation of which is discussed in the previous subsection.

Tables 6, 7, and 8 contain the metrics for single-threaded, double-threaded, and triple-threaded implementations operating upon 25 SNPs. The TTE decreases as the number of threads increases. In comparison to the single-threaded implementation, double-threading causes an average of 1.6 speedup while triple-threading can cause an average of 2.3 speedup. AET, ADT, and ART all increase as the number of threads increases. This is to be expected as a result of executing computations in parallel.

In light of this observation, the proceeding discussions use the triple-threaded implementation because while the AET, ADT, and ART increase, the TTE still is minimum after parallelization of computation.

Statistic	TTE	AET	ADT	ART
χ^2 test statistic	233 s	344 ms	446 ms	7974 ms
allelic odds ratio	233 s	344 ms	446 ms	7970 ms
Hardy-Weinberg equilibrium	280 s	343 ms	442 ms	9820 ms
minor allele frequency	174 s	345 ms	376 ms	5676 ms
linkage disequilibrium	564 s	714 ms	671 ms	22704 ms

Table 6: Timings for one thread

Statistic	TTE	AET	ADT	ART
χ^2 test statistic	146 s	367 ms	482 ms	8004 ms
allelic odds ratio	146 s	364 ms	488 ms	8009 ms
Hardy-Weinberg equilibrium	173 s	376 ms	484 ms	9859 ms
minor allele frequency	118 s	407 ms	420 ms	5712 ms
linkage disequilibrium	306 s	771 ms	746 ms	22790 ms

Table 7: Timings for two threads

Statistic	TTE	AET	ADT	ART
χ^2 test statistic	101 s	395 ms	501 ms	8056 ms
allelic odds ratio	102 s	402 ms	509 ms	8080 ms
Hardy-Weinberg equilibrium	122 s	449 ms	541 ms	9920 ms
minor allele frequency	181 s	430 ms	450 ms	5761 ms
linkage disequilibrium	210 s	842 ms	751 ms	23223 ms

Table 8: Timings for three threads

Tables 9, 10, 11, 12, 13, and 14 contain the timing metrics for the computation of each of the statistics for 10, 25, 50, 100, and 500 SNPs, each with three replications.

In the tabulation of these results, AET, ADT, and ART all remain around the same range regardless of how many SNPs are being processed in any of the computations. The TTE is observed to increase linearly with the number of SNPs, as expected, because despite the use of three threads, the computation eventually leads into a sequence of three computations occurring in parallel.

With all timings considered, we observe that the AET, ADT, and ART change with the number of threads used, increasing as the number of threads do and the TTE decreases as the number or threads increase and increases as the number of SNPs increase.

	TTE	AET	ADT	ART
10 SNPS	47 s	405 ms	510 ms	8157 ms
	46 s	386 ms	517 ms	8066 ms
	46 s	407 ms	497 ms	8134 ms
25 SNPS	101 s	387 ms	477 ms	8030 ms
	101 s	387 ms	483 ms	8038 ms
	103 s	458 ms	540 ms	8161 ms
50 SNPS	195 s	409 ms	498 ms	8068 ms
	196 s	417 ms	496 ms	8042 ms
	197 s	413 ms	515 ms	8095 ms
100 SNPS	405 s	412 ms	531 ms	8064 ms
	408 s	426 ms	512 ms	8104 ms
	407 s	412 ms	495 ms	8072 ms
500 SNPS	1907 s	401 ms	530 ms	8050 ms
	1908 s	389 ms	502 ms	8023 ms
	1910 s	395 ms	510 ms	8022 ms

Table 9: Timings for the computation of χ^2 test statistic

	TTE	AET	ADT	ART
10 SNPS	46 s	412 ms	519 ms	8040 ms
	46 s	423 ms	503 ms	8109 ms
	46 s	396 ms	546 ms	8119 ms
25 SNPS	103 s	422 ms	508 ms	8098 ms
	102 s	403 ms	491 ms	8047 ms
	102 s	409 ms	508 ms	8089 ms
50 SNPS	195 s	382 ms	506 ms	8020 ms
	198 s	454 ms	533 ms	8126 ms
	196 s	423 ms	509 ms	8086 ms
100 SNPS	406 s	417 ms	530 ms	8083 ms
	408 s	409 ms	515 ms	8051 ms
	405 s	406 ms	501 ms	8062 ms
500 SNPS	1908 s	398 ms	504 ms	8000 ms
	1911 s	404 ms	519 ms	8029 ms
	1911 s	413 ms	527 ms	8073 ms

Table 10: Timings for the computation of allelic odds ratio

	TTE	AET	ADT	ART
10 SNPS	54 s	449 ms	539 ms	9986 ms
	54 s	449 ms	500 ms	9985 ms
	54 s	456 ms	534 ms	9996 ms
25 SNPS	123 s	460 ms	559 ms	10009 ms
	121 s	433 ms	506 ms	9866 ms
	121 s	444 ms	508 ms	9920 ms
50 SNPS	228 s	424 ms	506 ms	9920 ms
	227 s	439 ms	486 ms	9863 ms
	229 s	429 ms	512 ms	9912 ms
100 SNPS	451 s	429 ms	485 ms	9850 ms
	454 s	424 ms	502 ms	9870 ms
	451 s	411 ms	515 ms	9960 ms
500 SNPS	2283 s	417 ms	522 ms	9876 ms
	2279 s	445 ms	519 ms	9868 ms
	2276 s	440 ms	530 ms	9855 ms

Table 11: Timings for the computation of Hardy-Weinberg equilibrium

	TTE	AET	ADT	ART
10 SNPS	37 s	420 ms	482 ms	5748 ms
	36 s	407 ms	429 ms	5715 ms
	36 s	401 ms	428 ms	5729 ms
25 SNPS	81 s	415 ms	427 ms	5719 ms
	83 s	462 ms	471 ms	5754 ms
	81 s	417 ms	430 ms	5710 ms
50 SNPS	152 s	404 ms	401 ms	5750 ms
	154 s	444 ms	434 ms	5714 ms
	153 s	404 ms	422 ms	5693 ms
100 SNPS	303 s	398 ms	432 ms	5731 ms
	302 s	418 ms	430 ms	5707 ms
	303 s	438 ms	428 ms	5716 ms
500 SNPS	1482 s	406 ms	395 ms	5701 ms
	1484 s	424 ms	425 ms	5738 ms
	1484 s	428 ms	422 ms	5711 ms

Table 12: Timings for the computation of minor allele frequency

	TTE	AET	ADT	ART
10 SNPS	79 s	812 ms	774 ms	22015 ms
	79 s	803 ms	762 ms	23059 ms
	79 s	854 ms	741 ms	22025 ms
25 SNPS	208 s	836 ms	777 ms	22493 ms
	207 s	836 ms	717 ms	22842 ms
	208 s	841 ms	729 ms	23031 ms
50 SNPS	433 s	810 ms	744 ms	22316 ms
	435 s	820 ms	769 ms	22436 ms
	436 s	818 ms	732 ms	23044 ms
100 SNPS	862 s	822 ms	778 ms	22289 ms
	857 s	808 ms	732 ms	22261 ms
	862 s	840 ms	749 ms	22342 ms
500 SNPS	4262 s	810 ms	732 ms	22868 ms
	4263 s	807 ms	759 ms	22871 ms
	4261 s	812 ms	752 ms	21913 ms

Table 13: Timings for the computation of linkage disequilibrium

	TTE	AET	ADT	ART
10 SNPS	17 s	10 ms	4 ms	348 ms
	17 s	10 ms	4 ms	348 ms
	17 s	10 ms	4 ms	348 ms
25 SNPS	42 s	19 ms	4 ms	851 ms
	42 s	19 ms	4 ms	851 ms
	42 s	19 ms	4 ms	851 ms
50 SNPS	83 s	34 ms	4 ms	1690 ms
	83 s	34 ms	4 ms	1690 ms
	83 s	34 ms	4 ms	1691 ms
100 SNPS	165 s	64 ms	4 ms	3369 ms
	165 s	64 ms	5 ms	3370 ms
	165 s	64 ms	4 ms	3369 ms
500 SNPS	822 s	314 ms	4 ms	16796 ms
	821 s	314 ms	4 ms	16796 ms
	820 s	314 ms	4 ms	16795 ms

Table 14: Timings for the computation of heterozygosity rate

VII. Conclusions

The tool developed in the completion of this study securely computes the χ^2 test statistic, allelic odds ratio, Hardy-Weinberg equilibrium, minor allele frequency, linkage disequilibrium, and heterozygosity rate of SNP data used in genome-wide association studies using fully homomorphic encryption.

It is comprised of a server and a client application which communicate using Windows Sockets, implements homomorphic encryption and related arithmetic operations using Simple Encrypted Arithmetic Library (SEAL), and can be accessed by the user through an interface developed using .NET.

As such, GWAS computations can now be outsourced to an external computing resource such as the cloud to maximize its computing power. The threat to security posed by uploading the data into a possibly untrusted resource is solved by homomorphic encryption which ensures the secure computation of the data by preserving its utility in mathematical operations, thus eliminating the need to decrypt the data in order for it to be meaningful in computation.

VIII. Recommendations

The main point of improvement to consider when using fully homomorphic encryption is its execution time. In doing so, one might look into the use more efficient encryption schemes that allow for the same or greater computing capabilities.

Other enhancements can include the addition of more statistics that can be used for GWAS such as genotype-based χ^2 test statistics and extension of functionality by the handling of non-biallelic SNPs. An approach to the second objective might begin in representing a SNP with additional polynomials, accounting for each of the additional heterozygous genotypes and all but one homozygous genotypes for the alternate allele.

IX. Bibliography

[1] W. S. Bush and J. H. Moore, "Genome-wide association studies," *PLoS Computational Biology*, vol. 8, no. 12, p. e1002822, 2012.

[2] P. M. Visscher, M. A. Brown, M. I. McCarthy, and J. Yang, "Five years of gwas discovery," *The American Journal of Human Genetics*, vol. 90, no. 1, pp. 7–24, 2012.

[3] P. G. C. C. Committee *et al.*, "Genomewide association studies: history, rationale, and prospects for psychiatric disorders," *American Journal of Psychiatry*, 2009.

[4] G. M. Clarke, C. A. Anderson, F. H. Pettersson, L. R. Cardon, A. P. Morris, and K. T. Zondervan, "Basic statistical analysis in genetic case-control studies," *Nature Protocols*, vol. 6, no. 2, pp. 121–133, 2011.

[5] B. E. Stranger, E. A. Stahl, and T. Raj, "Progress and promise of genome-wide association studies for human complex trait genetics," *Genetics*, vol. 187, no. 2, pp. 367–383, 2011.

[6] C. A. Anderson, F. H. Pettersson, G. M. Clarke, L. R. Cardon, A. P. Morris, and K. T. Zondervan, "Data quality control in genetic case-control association studies," *Nature protocols*, vol. 5, no. 9, pp. 1564–1573, 2010.

[7] C. C. Laurie, K. F. Doheny, D. B. Mirel, E. W. Pugh, L. J. Bierut, T. Bhangale, F. Boehm, N. E. Caporaso, M. C. Cornelis, H. J. Edenberg, *et al.*, "Quality control and quality assurance in genotypic data for genome-wide association studies," *Genetic Epidemiology*, vol. 34, no. 6, pp. 591–602, 2010.

[8] S. Turner, L. L. Armstrong, Y. Bradford, C. S. Carlson, D. C. Crawford, A. T. Crenshaw, M. Andrade, K. F. Doheny, J. L. Haines, G. Hayes, *et al.*,

"Quality control procedures for genome-wide association studies," *Current Protocols in Human Genetics*, pp. 1–19, 2011.

[9] "Creating a global alliance to enable responsible sharing of genomic and clinical data." http://genomicsandhealth.org/about-the-global-alliance/key-documents/white-paper-creating-global-alliance-enable-responsible-shar, 2013. Accessed: 2016-11-02.

[10] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo, "A new way to protect privacy in large-scale genome-wide association studies.," *Bioinformatics (Oxford, England)*, vol. 29, pp. 886–93, apr 2013.

[11] B. Malin and L. Sweeney, "Inferring genotype from clinical phenotype through a knowledge based algorithm," in *Proceedings of the Pacific Symposium on Biocomputing*, pp. 41–52, 2001.

[12] M. Phillips, B. M. Knoppers, and Y. Joly, "Seeking a" race to the top" in genomic cloud privacy?," in *Security and Privacy Workshops (SPW), 2015 IEEE*, pp. 65–69, IEEE, 2015.

[13] Z. Lin, A. B. Owen, and R. B. Altman, "Genomic research and human subject privacy," *Science*, vol. 305, no. 5681, 2004.

[14] M. Humbert, E. Ayday, J.-P. Hubaux, and A. 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*, pp. 1141–1152, ACM, 2013.

[15] L. A. Hindorff, P. Sethupathy, H. A. Junkins, E. M. Ramos, J. P. Mehta, F. S. Collins, and T. A. Manolio, "Potential etiologic and functional implications of genome-wide association loci for human diseases and traits," *Proceedings of the National Academy of Sciences*, vol. 106, no. 23, pp. 9362–9367, 2009.

[16] M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich, "Identifying personal genomes by surname inference," *Science*, vol. 339, no. 6117, pp. 321–324, 2013.

[17] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays," *PLoS Genetics*, vol. 4, no. 8, p. e1000167, 2008.

[18] D. W. Craig, R. M. Goor, Z. Wang, J. Paschall, J. Ostell, M. Feolo, S. T. Sherry, and T. A. Manolio, "Assessing and managing risk when sharing aggregate genetic variant data," *Nature Reviews Genetics*, vol. 12, no. 10, pp. 730–736, 2011.

[19] R. Cai, Z. Hao, M. Winslett, X. Xiao, Y. Yang, Z. Zhang, and S. Zhou, "Deterministic identification of specific individuals from GWAS results," *Bioinformatics*, vol. 31, no. 11, pp. 1701–1707, 2015.

[20] S. S. Shringarpure and C. D. Bustamante, "Privacy risks from genomic data-sharing beacons," *The American Journal of Human Genetics*, vol. 97, no. 5, pp. 631–646, 2015.

[21] C. A. Cassa, R. A. Miller, and K. D. Mandl, "A novel, privacy-preserving cryptographic approach for sharing sequencing data," *Journal of the American Medical Informatics Association*, vol. 20, no. 1, pp. 69–76, 2013.

[22] Z. Huang, E. Ayday, J. Fellay, J. P. Hubaux, and A. Juels, "GenoGuard: Protecting genomic data against brute-force attacks," *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2015-July, pp. 447–462, 2015.

[23] E. Ayday, J. L. Raisaro, J. Rougemont, and J.-P. Hubaux, "Protecting and evaluating genomic privacy in medical tests and personalized medicine," *Pro-*

ceedings of the 12th ACM workshop on Workshop on privacy in the electronic society - WPES '13, pp. 95–106, 2013.

[24] L. Barman, M.-T. Elgraini, J. L. Raisaro, J.-P. Hubaux, and E. Ayday, "Privacy threats and practical solutions for genetic risk tests," in *Security and Privacy Workshops (SPW), 2015 IEEE*, pp. 27–31, IEEE, 2015.

[25] Y. Zhao, X. Wang, and H. Tang, "Secure genomic computation through site-wise encryption," *AMIA Summits on Translational Science Proceedings*, vol. 2015, pp. 227–231, 2015.

[26] S. Simmons, C. Sahinalp, and B. Berger, "Enabling privacy-preserving GWASs in heterogeneous human populations," *Cell Systems*, vol. 3, pp. 54–61, jul 2016.

[27] F. Yu, S. E. Fienberg, A. B. Slavković, and C. Uhler, "Scalable privacy-preserving data sharing methodology for genome-wide association studies," *Journal of Biomedical Informatics*, vol. 50, pp. 133–141, 2014.

[28] F. Tramèr, Z. Huang, J.-P. Hubaux, and E. Ayday, "Differential privacy with bounded priors: reconciling utility and privacy in genome-wide association studies," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1286–1297, ACM, 2015.

[29] Y. Zhang, M. Blanton, and G. Almashaqbeh, "Secure distributed genome analysis for GWAS and sequence comparison computation," *BMC Medical Informatics and Decision Making*, vol. 15, no. Suppl 5, p. S4, 2015.

[30] S. Wang, Y. Zhang, W. Dai, K. Lauter, M. Kim, Y. Tang, H. Xiong, and X. Jiang, "Healer: Homomorphic computation of exact logistic regression for secure rare disease variants analysis in gwas," *Bioinformatics*, vol. 32, no. 2, pp. 211–218, 2016.

[31] K. Lauter, A. López-Alt, and M. Naehrig, "Private computation on encrypted genomic data," in *International Conference on Cryptology and Information Security in Latin America*, pp. 3–27, Springer, 2014.

[32] W.-J. Lu, Y. Yamada, and J. Sakuma, "Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption," *BMC Medical Informatics and Decision Making*, vol. 15, no. Suppl 5, pp. 1–8, 2015.

[33] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," *BMC Medical Informatics and Decision Making*, vol. 15, no. Suppl 5, p. S3, 2015.

[34] Y. Zhang, W. Dai, X. Jiang, H. Xiong, and S. Wang, "Foresee: Fully outsourced secure genome study based on homomorphic encryption," *BMC medical informatics and decision making*, vol. 15, no. Suppl 5, p. S5, 2015.

[35] A. López-Alt and M. Naehrig, "Large integer plaintexts in ring-based fully homomorphic encryption." In preparation, 2014.

[36] S. Halevi and V. Shoup, "Algorithms in HElib," in *International Cryptology Conference*, pp. 554–571, Springer, 2014.

[37] R. P. Kim Laine, Hao Chen, "Simple encrypted arithmetic library - seal (v2.1)," tech. rep., September 2016.

[38] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption.," *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[39] J. F. Brookfield, "Q&A: promise and pitfalls of genome-wide association studies," *BMC Biology*, vol. 8, no. 1, p. 1, 2010.

[40] M. E. Tabangin, J. G. Woo, and L. J. Martin, "The effect of minor allele frequency on the likelihood of obtaining false positives," in *BMC Proceedings*, vol. 3, p. S41, BioMed Central Ltd, 2009.

[41] J. E. Wigginton, D. J. Cutler, and G. R. Abecasis, “A note on exact tests of hardy-weinberg equilibrium,” *The American Journal of Human Genetics*, vol. 76, no. 5, pp. 887–893, 2005.

[42] G. Salanti, G. Amountza, E. E. Ntzani, and J. P. Ioannidis, “Hardy–weinberg equilibrium in genetic association studies: an empirical evaluation of reporting, deviations, and power,” *European Journal of Human Genetics*, vol. 13, no. 7, pp. 840–848, 2005.

[43] M. Slatkin, “Linkage disequilibrium—understanding the evolutionary past and mapping the medical future,” *Nature Reviews Genetics*, vol. 9, no. 6, pp. 477–485, 2008.

[44] A. Pluzhnikov, J. E. Below, A. Konkashbaev, A. Tikhomirov, E. Kistner-Griffin, C. A. Roe, D. L. Nicolae, and N. J. Cox, “Spoiling the whole bunch: quality control aimed at preserving the integrity of high-throughput genotyping,” *The American Journal of Human Genetics*, vol. 87, no. 1, pp. 123–128, 2010.

[45] L. J. Aslett, P. M. Esperança, and C. C. Holmes, “A review of homomorphic encryption and software tools for encrypted statistical machine learning.” 2015.

[46] L. Morris, “Analysis of partially and fully homomorphic encryption.” `http://www.liammorris.com/crypto2/Homomorphic%20Encryption%20Paper.pdf`, 2013. Accessed: 2016-11-02.

X. Appendix

A. Source Code

A..1 Server module

Server module: GWASServer.cpp

```
// GWASServer.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "snpserver.h"

using namespace seal;
using namespace std;

WSADATA wsa;
SOCKET s;
DWORD threadId;

bool initializeWinsock();
bool createListener();
bool acceptRequests();
DWORD WINAPI doGwas(void* sd);

int main()
{
  bool winsockStatus, listenerStatus, acceptStatus;
  //Initialize winsock]
  do {
    winsockStatus = initializeWinsock();
  } while (!winsockStatus);

  //Create listener
  do {
    listenerStatus = createListener();
  } while (!listenerStatus);

  //Accept requests
  do {
    acceptStatus = acceptRequests();
  } while (!acceptStatus);

  WSACleanup();

  // Wait for ENTER before closing screen.
  cout << "Press ENTER to exit" << endl;
  char ignore;
  cin.get(ignore);
  return 0;
}

bool initializeWinsock()
{
  cout << "Initialising Winsock..." << endl;
  if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
    cout << "Failed. Error Code : " << WSAGetLastError() << endl;
    return false;
  }
  cout << "Initialised." << endl;
  return true;
}

bool createListener() {
  //Get config
  string line;
  ifstream configInputStream("GWASServer.config", ifstream::in);
  int port = 0;
```

```
    while (getline(configInputStream, line) && port == 0) {
      port = stoi(line);
    }
    configInputStream.close();

    //Create socket
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
      cout << "Could not create socket : " << WSAGetLastError() << endl;
      return false;
    }

    //Create server
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);

    //Bind socket
    if (::bind(s, (LPSOCKADDR)&addr, sizeof(addr)) == SOCKET_ERROR) {
      cout << "Could not bind socket" << endl;
      return false;
    }

    cout << "Socket created." << endl;
    return true;
  }

  bool acceptRequests() {
    try {
      while (true) {
        listen(s, SOMAXCONN);

        //Accept a client socket
        SOCKET client = accept(s, NULL, NULL);
        if (client == INVALID_SOCKET) {
          printf("accept failed with error: %d\n", WSAGetLastError());
        }
        else {
          CloseHandle(CreateThread(NULL, 0, doGwas, (void*)client, 0, &threadId));
        }
      }
    }
    catch (...) {
      return false;
    }
    return false;
  }

DWORD WINAPI doGwas(void* sd) {
  SOCKET client = (SOCKET)sd;

    //Receive transaction header
    vector<char> recvbuf(DEFAULT_BUFLEN);
    int recvbuflen = DEFAULT_BUFLEN;
    int has_new = 0;

    if (client == INVALID_SOCKET) {
      return 1;
    }
    else {
      has_new = recv(client, recvbuf.data(), recvbuflen, 0);
    }

    vector<string> header = split(string(recvbuf.data(), has_new), '|');
    if (header.size() < 2) {
      closesocket(client);
      return 1;
    }

    string mode = header[0];
    int size = stoi(header[1]);

    //Contingency table only or contingency table with HWE
```

```
if (mode == "CO" || mode == "CH" || mode == "CM") {
  int mod = max((int)pow(2, ceil(log2(size)) + 2), 2048);

  //Receiving ciphertext strings
  vector<string> cipherstrs = recvNCiphertexts(3, client, (int)pow(2, ceil(log2
  (mod)) + 4) + 28);

  if (cipherstrs.size() < 3) {
    closesocket(client);
    cout << "Error in receiving ciphertexts" << endl;
    return 1;
  }

  try {
    //Loading ciphertext objects
    vector<BigPolyArray> ciphertexts = loadCiphertexts(cipherstrs);

    if (ciphertexts.size() < 3) {
      closesocket(client);
      return 1;
    }

    //Setting evaluator parameters
    EncryptionParameters parms;
    parms.poly_modulus() = "1x^" + to_string(mod) + " + 1";
    parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(
  mod);
    parms.plain_modulus() = (int)pow(2, ceil(log2(size)) + 2);

    BigPoly onePoly = BigPoly(kPolyNterms(1, size));

    //Performing homomorphic computations
    cout << "Performing arithmetic on encrypted numbers..." << endl;
    Evaluator evaluator(parms);
    vector<BigPolyArray> ans;

    BigPolyArray encryptedsum = evaluator.add(ciphertexts[0], ciphertexts[1]);

    if (mode == "CO") {
      BigPolyArray encryptedacase = evaluator.multiply(encryptedsum,
  ciphertexts[2]);
      ans.push_back(encryptedacase);
    }

    BigPolyArray encrypteda = evaluator.multiply_plain(encryptedsum, onePoly);
    ans.push_back(encrypteda);

    if (mode == "CH") {
      BigPolyArray encryptedaa = evaluator.multiply_plain(ciphertexts[0],
  onePoly);
      ans.push_back(encryptedaa);

      BigPolyArray encryptedab = evaluator.multiply_plain(ciphertexts[1],
  onePoly);
      ans.push_back(encryptedab);
    }

    //Sending ciphertexts
    sendCiphertexts(ans, client);
  }
  catch (...) {
    closesocket(client);
    cout << "Error in homomorphic computation" << endl;
    return 1;
  }
}
else if (mode == "HR") {
  //Check header
  if (header.size() < 3) {
    closesocket(client);
    return 1;
  }
```

```
    int mod = max((int)pow(2, ceil(log2(stoi(header[2]))) + 2), 1024);

    //Receiving ciphertext strings
    vector<string> cipherstrs = recvNCiphertexts(size, client, (int)pow(2, ceil(
    log2(mod)) + 4) + 28);

    if (cipherstrs.size() < size) {
      closesocket(client);
      cout << "Error in receiving ciphertexts" << endl;
      return 1;
    }

    try {
      //Loading ciphertext objects
      vector<BigPolyArray> ciphertexts = loadCiphertexts(cipherstrs);

      if (ciphertexts.size() < size) {
        closesocket(client);
        return 1;
      }

      //Setting evaluator parameters
      EncryptionParameters parms;
      parms.poly_modulus() = "1x^" + to_string(mod) + " + 1";
      parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(
    mod);
      parms.plain_modulus() = (int)pow(2, ceil(log2(size)) + 1);

      //Performing homomorphic computations
      cout << "Performing arithmetic on encrypted numbers..." << endl;
      Evaluator evaluator(parms);
      BigPolyArray sum = ciphertexts[0];
      ciphertexts.erase(ciphertexts.begin());

      for (BigPolyArray ciphertext : ciphertexts) {
        sum = evaluator.add(sum, ciphertext);
      }

      //Sending one ciphertext
      vector<BigPolyArray> ans = vector<BigPolyArray>();
      ans.push_back(sum);
      sendCiphertexts(ans, client);
    }
    catch (...) {
      closesocket(client);
      cout << "Error in homomorphic computation" << endl;
      return 1;
    }
  }
  else if (mode == "LD") {
    int mod = max((int)pow(2, ceil(log2(size)) + 3), 2048);

    //Receiving ciphertext strings
    vector<string> cipherstrs = recvNCiphertexts(6, client, (int)pow(2, ceil(log2
    (mod)) + 4) + 28);

    if (cipherstrs.size() < 6) {
      closesocket(client);
      cout << "Error in receiving ciphertexts" << endl;
      return 1;
    }

    try {
      //Loading ciphertext objects
      vector<BigPolyArray> ciphertexts = loadCiphertexts(cipherstrs);

      if (ciphertexts.size() < 6) {
        closesocket(client);
        return 1;
      }

      //Setting evaluator parameters
      EncryptionParameters parms;
      parms.poly_modulus() = "1x^" + to_string(mod) + " + 1";
      parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(
    mod);
      parms.plain_modulus() = (int)pow(2, ceil(log2(size)) + 4);

      BigPoly onePoly = BigPoly(kPolyNterms(1, size));

      //Performing homomorphic computations
      cout << "Performing arithmetic on encrypted numbers ..." << endl;
      Evaluator evaluator(parms);
      vector<BigPolyArray> ans;

      ans.push_back(evaluator.multiply(ciphertexts[0], ciphertexts[4]));
      ans.push_back(evaluator.multiply(ciphertexts[1], ciphertexts[4]));
      ans.push_back(evaluator.multiply(ciphertexts[0], ciphertexts[5]));
      ans.push_back(evaluator.multiply(ciphertexts[1], ciphertexts[5]));

      BigPolyArray encryptedsum1 = evaluator.add(ciphertexts[0], ciphertexts[1]);
      ans.push_back(evaluator.multiply_plain(encryptedsum1, onePoly));

      BigPolyArray encryptedsum2 = evaluator.add(ciphertexts[2], ciphertexts[3]);
      ans.push_back(evaluator.multiply_plain(encryptedsum2, onePoly));

      //Sending ciphertexts
      sendCiphertexts(ans, client);
    }
    catch (...) {
      closesocket(client);
      cout << "Error in homomorphic computation" << endl;
      return 1;
    }
  }
  closesocket(client);
  return 0;
}
```

A..2 Client module

CaseControlTable.cpp

```
#include "stdafx.h"
#include "seal.h"
#include "CaseControlTable.h"

using namespace std;
using namespace seal;

CaseControlTable::CaseControlTable(Snp cases, Snp controls, SnpSocket sock,
    ModeType mode):snp(cases), s(sock), error(false), cases(cases.getSize()), a
    (0), acase(0), naa(0), nab(0) {
  if (mode == ModeType::CHISQ || mode == ModeType::AOR) {
    snp.add(controls);
  }

  //Encoding of SNP
  BigPoly homoPolyFw = BigPoly(this->snp.getHomoPolyFw());
  BigPoly heteroPolyFw = BigPoly(this->snp.getHeteroPolyFw());
  BigPoly phenoPolyBw = BigPoly(this->snp.getPhenotypePolyBw());

  //Setting encryption parameters
  int poly_mod = max((int)pow(2, ceil(log2(this->snp.getSize())) + 2),2048);
  int plain_mod = (int)pow(2, ceil(log2(this->snp.getSize())) + 2);
  //cout << "Modulus: " << poly_mod << "," << plain_mod << endl;

  EncryptionParameters parms;
  parms.poly_modulus() = "1x^" + to_string(poly_mod) + " + 1";
  parms.coeff_modulus() = 1 << 10;
  parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(
    poly_mod);
  parms.plain_modulus() = plain_mod;
```

```
//Generating keys
//cout << "Generating keys ..." << endl;
KeyGenerator generator(parms);
generator.generate();
BigPolyArray public_key = generator.public_key();
BigPoly secret_key = generator.secret_key();

//Timestamp 1
timestamps.push_back(std::chrono::steady_clock::now());

//Encrypting values
//cout << "Encrypting values..." << endl;
Encryptor encryptor(parms, public_key);

//Timestamp 2
timestamps.push_back(std::chrono::steady_clock::now());

vector<BigPolyArray> ciphertexts;
ciphertexts.push_back(encryptor.encrypt(homoPolyFw));
ciphertexts.push_back(encryptor.encrypt(heteroPolyFw));
ciphertexts.push_back(encryptor.encrypt(phenoPolyBw));

//Connect remotely
boolean isStarted = false;
int attempts = 0;
do {
  isStarted = this->s.start();
  attempts++;
} while (!isStarted && attempts < MAX_ATTEMPT);

if (!isStarted) {
  error = true;
  return;
}

//Sending computation type to server
int n = 0;
if (mode == ModeType::CHISQ || mode == ModeType::AOR) {
  this->s.sendMessage("CO|" + to_string(this->snp.getSize()));
  n = 2;
}
else if (mode == ModeType::HWE) {
  this->s.sendMessage("CH|" + to_string(this->snp.getSize()));
  n = 3;
}
else if (mode == ModeType::MAF) {
  this->s.sendMessage("CM|" + to_string(this->snp.getSize()));
  n = 1;
}

//Timestamp 3
timestamps.push_back(std::chrono::steady_clock::now());

//Sending data ciphertexts and receiving answer ciphertexts
vector<string> ans = vector<string>();
if (this->s.sendCiphertexts(ciphertexts)) {
  ans = this->s.recvNCiphertexts(n);

  //Timestamp 4
  timestamps.push_back(std::chrono::steady_clock::now());

  //Closing socket
  this->s.close();

  if (ans.size() == n) {
    //Timestamp 5
    timestamps.push_back(std::chrono::steady_clock::now());

    //Decrypting ciphertext
    //cout << "Decrypting results ..." << endl;
    vector<double> compute;
    Decryptor decryptor(parms, secret_key);
```

```
        for (string str : ans) {
          BigPolyArray polyarr;
          stringstream in = stringstream(str);
          polyarr.load(in);

          BigPoly decrypted = decryptor.decrypt(polyarr);
          compute.push_back(decrypted[this->snp.getSize()].to_double());
        }

        //Timestamp 6
        timestamps.push_back(std::chrono::steady_clock::now());

        //Saving results
        if (mode == ModeType::CHISQ || mode == ModeType::AOR) {
          this->acase = compute[0];
          this->a = compute[1];
        }
        else if (mode == ModeType::HWE) {
          this->a = compute[0];
          this->naa = compute[1] / 2;
          this->nab = compute[2];
        }
        else if (mode == ModeType::MAF) {
          this->a = compute[0];
        }
      }
      else {
        error = true;
        //cout << "Insufficient number of ciphertexts received from server.
      Terminating..." << endl;
        this->s.close();
      }
    }
    else {
      error = true;

      //Closing socket
      //cout << "Send error. Terminating..." << endl;
      this->s.close();
    }
  }

  CaseControlTable::~CaseControlTable() {}

  double CaseControlTable::chisq() {
    double m = snp.getSize();
    double bcase = 2 * cases - acase;
    double acontrol = a - acase;
    double b = 2 * m - a;
    double controls = m - cases;
    double bcontrol = b - bcase;

    return 2 * m * pow((bcase*(acontrol + bcontrol)) - (bcontrol*(acase + bcase)),
      2) / (cases*controls*a*b);
  }

  double CaseControlTable::aor() {
    double bcase = 2 * cases - acase;
    double acontrol = a - acase;
    double b = 2 * snp.getSize() - a;
    double bcontrol = b - bcase;

    return (acase*bcontrol) / (acontrol*bcase);
  }

  double CaseControlTable::maf() {
    return min(a, 2 * snp.getSize() - a) / (2 * snp.getSize());
  }

  double CaseControlTable::hwe() {
    double eaa = (a*a) / (4 * snp.getSize());
    double eab = (2 * a*(2 * snp.getSize() - a)) / (4 * snp.getSize());
  double ebb = ((2 * snp.getSize() - a)*(2 * snp.getSize() - a)) / (4 * snp.
    getSize());
  double nbb = snp.getSize() - naa - nab;

  return pow(naa - eaa, 2) / eaa + pow(nab - eab, 2) / eab + pow(nbb - ebb, 2) /
    ebb;
}


chrono::milliseconds CaseControlTable::encryptionTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[1] -
    timestamps[0]);
  return (chrono::seconds)0;
}

chrono::milliseconds CaseControlTable::remoteTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[3] -
    timestamps[2]);
  return (chrono::seconds)0;
}

chrono::milliseconds CaseControlTable::decryptionTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[5] -
    timestamps[4]);
  return (chrono::seconds)0;
}

boolean CaseControlTable::isError() {
  return error;
}
```

LinkageDisequilibrium.cpp

```
#include "stdafx.h"
#include "seal.h"
#include "LinkageDisequilibrium.h"

using namespace std;
using namespace seal;

LinkageDisequilibrium::LinkageDisequilibrium(Snp firstSnp, Snp secondSnp,
    SnpSocket sock) :snp1(firstSnp), snp2(secondSnp), s(sock), error(false) {
  //Encoding of SNPs
  vector<BigPoly> polys;
  polys.push_back(BigPoly(snp1.getHomoPolyFw()));
  polys.push_back(BigPoly(snp1.getHeteroPolyFw()));
  polys.push_back(BigPoly(snp2.getHomoPolyFw()));
  polys.push_back(BigPoly(snp2.getHeteroPolyFw()));
  polys.push_back(BigPoly(snp2.getHomoPolyBw()));
  polys.push_back(BigPoly(snp2.getHeteroPolyBw()));

  //Setting encryption parameters
  int poly_mod = max((int)pow(2, ceil(log2(this->snp1.getSize())) + 3),2048);
  int plain_mod = (int)pow(2, ceil(log2(this->snp1.getSize())) + 4);
  //cout<< "Modulus: " << poly_mod << "," << plain_mod << endl;

  EncryptionParameters parms;
  parms.poly_modulus() = "1x^" + to_string(poly_mod) + " + 1";
  parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(
    poly_mod);
  parms.plain_modulus() = plain_mod;

  //Generating keys
  //cout<< "Generating keys ..." << endl;
  KeyGenerator generator(parms);
  generator.generate();
  BigPolyArray public_key = generator.public_key();
  BigPoly secret_key = generator.secret_key();

  //Timestamp 1
  timestamps.push_back(std::chrono::steady_clock::now());

  //Encrypting values
  //cout<< "Encrypting values..." << endl;
```

```
Encryptor encryptor(parms, public_key);
vector<BigPolyArray> ciphertexts;

for (BigPoly poly : polys) {
  ciphertexts.push_back(encryptor.encrypt(poly));
}

//Timestamp 2
timestamps.push_back(std::chrono::steady_clock::now());

//Connect remotely
boolean isStarted = false;
int attempts = 0;
do {
  isStarted = this->s.start();
  attempts++;
} while (!isStarted && attempts < MAX_ATTEMPT);

if (!isStarted) {
  error = true;
  return;
}

//Sending computation type to server
this->s.sendMessage("LD|" + to_string(this->snp1.getSize()));

//Timestamp 3
timestamps.push_back(std::chrono::steady_clock::now());

//Sending data ciphertexts and receiving answer ciphertexts
vector<string> ans = vector<string>();
if (this->s.sendCiphertexts(ciphertexts)) {
  ans = this->s.recvNCiphertexts(6);

  //Timestamp 4
  timestamps.push_back(std::chrono::steady_clock::now());

  //Closing socket
  this->s.close();

  if (ans.size() == 6) {
    //Timestamp 5
    timestamps.push_back(std::chrono::steady_clock::now());

    //Decrypting ciphertext
    //cout<< "Decrypting results ..." << endl;
    vector<double> compute;
    Decryptor decryptor(parms, secret_key);

    for (string str : ans) {
      BigPolyArray polyarr;
      stringstream in = stringstream(str);
      polyarr.load(in);

      BigPoly decrypted = decryptor.decrypt(polyarr);
      compute.push_back(decrypted[snp1.getSize() - 1].to_double());
      compute.push_back(decrypted[snp1.getSize()].to_double());
    }

    //Timestamp 6
    timestamps.push_back(std::chrono::steady_clock::now());

    homohomo = compute[0] / 4;
    heterohomo = compute[2] / 2;
    homohetero = compute[4] / 2;
    heterohetero = compute[6];
    refAllele1 = compute[9];
    refAllele2 = compute[11];
  }
  else {
    error = true;
    //cout<< "Insufficient number of ciphertexts received from server.
  Terminating..." << endl;
```

```
      }
    }
    else {
      error = true;

      //Closing socket
      //cout<< "Send error. Terminating..." << endl;
      this->s.close();
    }
  }

LinkageDisequilibrium::~LinkageDisequilibrium() {}

double LinkageDisequilibrium::ld() {
  double pab = (2 * homohomo + heterohomo + homohetero) / (2 * snp1.getSize());
  double pa = (refAllele1 - heterohetero) / (2 * snp1.getSize());
  double pb = (refAllele2 - heterohetero) / (2 * snp1.getSize());

  double d = pab - pa * pb;
  double dmax = d > 0 ? min(pa*(1 - pb), (1 - pa)*pb) : min(pa*pb, (1 - pa)*(1 -
    pb));

  return d / dmax;
}

chrono::milliseconds LinkageDisequilibrium::encryptionTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[1] -
    timestamps[0]);
  return (chrono::seconds)0;
}

chrono::milliseconds LinkageDisequilibrium::remoteTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[3] -
    timestamps[2]);
  return (chrono::seconds)0;
}

chrono::milliseconds LinkageDisequilibrium::decryptionTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[5] -
    timestamps[4]);
  return (chrono::seconds)0;
}

boolean LinkageDisequilibrium::isError() {
  return error;
}
```

Heterozygosity.cpp

```
#include "stdafx.h"
#include "seal.h"
#include "Heterozygosity.h"

using namespace std;
using namespace seal;

Heterozygosity::Heterozygosity(vector<Snp> snparr, SnpSocket sock) : snps(snparr)
    , s(sock), error(false) {
  //Setting encryption parameters
  int poly_mod = max((int)pow(2, ceil(log2(this->snps[0].getSize())) + 2),1024);
  int plain_mod = (int)pow(2, ceil(log2(this->snps.size())) + 1);
  //cout << "Modulus: " << poly_mod << "," << plain_mod << endl;

  EncryptionParameters parms;
  parms.poly_modulus() = "1x^" + to_string(poly_mod) + " + 1";
  parms.coeff_modulus() = ChooserEvaluator::default_parameter_options().at(
    poly_mod);
  parms.plain_modulus() = plain_mod;

  //Generating keys
  //cout << "Generating keys ..." << endl;
  KeyGenerator generator(parms);
  generator.generate();
  BigPolyArray public_key = generator.public_key();
```

```
BigPoly secret_key = generator.secret_key();

//Timestamp 1
timestamps.push_back(std::chrono::steady_clock::now());

//Encrypting values
//cout << "Encrypting and sending values..." << endl;
Encryptor encryptor(parms, public_key);

vector<BigPolyArray> ciphertexts;
for (Snp snp : snps) {
  ciphertexts.push_back(encryptor.encrypt(snp.getHeteroPolyFw()));
}

//Timestamp 2
timestamps.push_back(std::chrono::steady_clock::now());

//Connect remotely
boolean isStarted = false;
int attempts = 0;
do {
  isStarted = this->s.start();
  attempts++;
} while (!isStarted && attempts < MAX_ATTEMPT);

if (!isStarted) {
  error = true;
  return;
}

//Sending computation type to server
this->s.sendMessage("HR|" + to_string(this->snps.size()) + "|" + to_string(this
  ->snps[0].getSize()));

//Timestamp 3
timestamps.push_back(std::chrono::steady_clock::now());

//Receiving one ciphertext
if (this->s.sendCiphertexts(ciphertexts)) {
  vector<string> ans = this->s.recvNCiphertexts(1);

  //Timestamp 4
  timestamps.push_back(std::chrono::steady_clock::now());

  //Closing socket
  this->s.close();

  if (ans.size() == 1) {
    //Timestamp 5
    timestamps.push_back(std::chrono::steady_clock::now());

    //Loading ciphertext
    BigPolyArray polyarr;
    stringstream in = stringstream(ans[0]);
    polyarr.load(in);

    //Decrypting ciphertext
    Decryptor decryptor(parms, secret_key);
    //cout << "Decrypting results ..." << endl;
    BigPoly decrypted = decryptor.decrypt(polyarr);

    //Timestamp 6
    timestamps.push_back(std::chrono::steady_clock::now());

    for (int ctr = 0; ctr < snps[0].getSize(); ctr++) {
      heterozygosities.push_back(decrypted[ctr].to_double());
    }
  }
  else {
    error = true;
    //cout << "Insufficient number of ciphertexts received from server.
  Terminating..." << endl;
  }
  }
  else {
    error = true;

    //Closing socket
    //cout << "Send error. Terminating..." << endl;
    this->s.close();
  }
}

Heterozygosity::~Heterozygosity() {}

double Heterozygosity::hr(int index) {
  return heterozygosities[index] / snps.size();
}

chrono::milliseconds Heterozygosity::encryptionTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[1] -
    timestamps[0]);
  return (chrono::seconds)0;
}

chrono::milliseconds Heterozygosity::remoteTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[3] -
    timestamps[2]);
  return (chrono::seconds)0;
}

chrono::milliseconds Heterozygosity::decryptionTime() {
  if (!error) return chrono::duration_cast<chrono::milliseconds>(timestamps[5] -
    timestamps[4]);
  return (chrono::seconds)0;
}

boolean Heterozygosity::isError() {
  return error;
}
```

XI. Acknowledgement

These are the people to whom I dedicate this work.

To my parents, who are my pillars, thank you for your unwavering support, for all your hardwork. Thank you for sharing my dreams and striving to make them come true as if they were your own.

To my brother, grandparents, and relatives, thank you for always having my back and egging me on through college. You took me under your wing and helped me spread my own.

To my adviser, Sir Richard Bryann Chua, you always push us to do our best. We have much to thank you for but nothing can be more important than the growth you helped spur in us, your advisees. Thank you for sharing to us your guidance and time.

To my friends, especially my blockmates, it is in those little moments we share together that I find the motivation to push further, and amidst everything, the joy in all we do.

To my best friend, to you, the world is always in vivid color; and every time I forget how beautiful the colors are, you remind me unfailingly. Thank you for always believing in me.

To God, thank you for blessing me with the acquaintance of these people. It is through them that I receive and give Your love.

I could not have done any of this without you. As words will never be enough to express my gratitude, I hope that the completion this work, one of the hardest things I ever had to do in my life, will show you how much you have helped me along the way. I share this success with you. Thank you so much.