Private Random Variate Sampling for Secure and Federated Polygenic Risk Scores

by

Derek Jia-Wen Yen

B.S. Computer Science and Engineering, Linguistics and Philosophy, MIT, 2022

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by:	Derek Jia-Wen Yen Department of Electrical Engineering and Computer Science January 19, 2024
Certified by:	Bonnie Berger Professor of Mathematics, Thesis Supervisor
Certified by:	Hyunghoon Cho Assistant Professor of Biomedical Informatics and Data Science Yale University, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Private Random Variate Sampling for Secure and Federated Polygenic Risk Scores

by

Derek Jia-Wen Yen

Submitted to the Department of Electrical Engineering and Computer Science
on January 19, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Polygenic risk scores (PRS) are used to quantify the additive effect of single nucleotide polymorphisms (SNPs) on an individual's genetic risk for developing a particular trait or condition. Collaborations between data centers are important for improving the statistical power and validity of PRS through larger, more genetically diverse datasets. However, owing to the privacy concerns inherent in genomic data, regulations restrict institutions' capacity to share data. Using cryptography, we present a secure and federated implementation of a Monte Carlo algorithm for PRS, enabling collaborations that respect data regulations. To implement a Monte Carlo algorithm in a privacy-preserving context, our work exhibits techniques for sampling random variates with cryptographically private parameters, which may be of independent interest.

Thesis supervisor: Bonnie Berger
Title: Professor of Mathematics

Thesis supervisor: Hyunghoon Cho
Title: Assistant Professor of Biomedical Informatics and Data Science, Yale University

Acknowledgments

Thanks to Manaswitha Edupalli and David Froelicher for their guidance both within and beyond the lab. This past year has been formative in my development as a researcher, and I owe that in no small part to their mentorship and advice.

Thanks to Professor Hyunghoon Cho and Professor Bonnie Berger for supervising my research. I am grateful to have been given the opportunity to pursue this work and will cherish my time in their groups.

Lastly, I thank my family and friends for supporting me throughout my MEng: you have been my anchor amidst waves.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Polygenic risk scores (PRS) are a promising technique to give patients a more informed picture of their relative risk for developing particular conditions, such as chronic diseases, based on their genetic variants. [1], [2] Collaborations between data centers to pool larger datasets improve the statistical power of the resulting PRS, but the sensitive nature of genomic/medical data has led to regulations on data centers' ability to legally share such data. Consequently, it can be difficult in practice to effect such collaborations.

Our research aims to demonstrate that this tension between privacy and research can be resolved through cryptography. Two particular cryptographic tools are useful for privacy-preserving collaborative computations: homomorphic encryption (HE) and multiparty computation (MPC). Both of these tools offer techniques for parties to jointly compute a function of their individual datasets without revealing any sensitive information to the other parties. Our work will thereby facilitate collaborations by allowing data centers to obtain the same results as though they had shared data while still respecting regulations.

In implementing a PRS algorithm with these privacy-preserving technologies, our work develops techniques for sampling random variates from distributions with cryptographically private parameters, which may be of interest for privacy-preserving computation beyond the biomedical context.

1.1 Problem definition

Cryptographic techniques offer a natural method to address the privacy concerns that arise in computations with genomic data. [3], [4] More broadly, privacy preserving multiparty computation has been developed for a variety of domains, including secure advertising and privacy-preserving machine learning. [5], [6] Our work contributes to a large and diverse literature of applications of cryptographic methods for collaboratively computing an output on private inputs; or outsourcing an expensive computation on sensitive data to a third party without revealing said data. Following prior work, we use a combination of two cryptographic techniques—homomorphic encryption and multiparty computation—to leverage the benefits of both techniques. [7]–[9]

We assume a *secure* and *federated* problem context, in which P parties holding disjoint sets of genomic data wish to collaborate and compute PRS scores over their shared data without revealing any sensitive information to the other parties. We assume that the parties can communicate with each other over a secure and authenticated channel (such as through the TLS protocol); and also that they have mutually selected a particular computing party, the "hub party," to be distinguished as the one party to perform certain special operations in algorithms where breaking symmetry is necessary. We also assume that there is a so called "party 0" to serve as a "trusted dealer" for MPC. Lastly, and crucially, we assume an "honest but curious" security model: the parties can be trusted to correctly follow the steps of the protocol, but will seek to learn the other parties' private values from all information that they observe over the course of communications. [10] The cryptographic tools we use will satisfy this security model.

Of the various recent algorithms for PRS, we implement PRS-CS by Ge et al. owing to its relevance and influence. [11] The PRS-CS algorithm performs a Bayesian regression through Gibbs sampling, a Markov Chain Monte Carlo method for sampling from a multivariate probability distribution. Implementing such an algorithm in our secure setting entails implementing its constituent random variable sampling functions in a cryptographically secure way. To our knowledge, this is an unexplored problem in privacy-preserving computation, and one which may be of independent interest owing to the prevalence of Monte Carlo algorithms.

1.2 Description of thesis structure

In the rest of this chapter, I will review the relevant genomics background for the problem context, and the two cryptographic tools used in our work.

In Chapter 2, I discuss our implementation of PRS-CS's sampling functions. I will review the PRS-CS algorithm, discuss some general considerations of working within the HE/MPC framework, then present our secure random variate sampling algorithms. As a primary contribution of this work is in broaching the problem of secure random variate sampling, I will emphasize the considerations used to achieve computational efficiency within the HE/MPC framework, which may provide insight on how to approach other probabilistic applications of privacy preserving multiparty computation. Discussion of the sampling algorithms will highlight the reasoning used to achieve computational efficiency within the HE/MPC framework, which may provide insight on how to approach other problems within the space of privacy preserving multiparty computation.

In Chapter 3, I will demonstrate the faithfulness of the secure sampling algorithms, discuss preliminary results on the runtime of our implementation, and denote avenues of future work. Lastly, the Appendix contains additional details about the theory underlying β sampling.

1.3 Genomics background

Single nucleotide polymorphisms (SNPs) are variant alleles in a population that differ by a single base pair from a reference genome. Through genome-wide association studies (GWAS), it is possible to identify SNPs correlated with particular phenotypes and assign each SNP an "effect size" $\hat{\beta}_j$ measuring the strength of its correlation with the expression of that phenotype. The effect sizes generated by GWAS can be further refined for polygenic risk scores (PRS), which quantify an individual's genetic risk for a phenotype based on the SNPs present in their genome. [1], [2], [12]

In standard PRS models, it is assumed that an individual's relative risk can be represented as a sum of the effect sizes of the SNPs that individual possesses: in other words, assuming that an individual's SNPs are independent from each other. A major consideration in generating PRS effect sizes is therefore accounting for linkage disequilibrium (LD) between SNPs, which is a measure of how correlated the inheritance of two SNPs is. [2] Accounting for LD effects is known to improve the accuracy of PRS. [13]

Furthermore, the accuracy of PRS is sensitive to the particular population used in the input GWAS. GWAS statistics and LD estimates can differ between populations of different ancestry groups and population structure, so PRS are most accurate when generated from data of a similar ancestry group as the patient. [1], [12], [14] This variation underscores the clinical importance of aggregating genomics datasets for larger and more diverse studies, as most prior genomics research has focused on European datasets and may not transfer to other populations. [1]

1.3.1 Notation and dataset

Here, we review the assumed problem context and the notation associated with the genomics datasets. Assume that computing parties indexed $p \in [P]$ each hold the genotype data for disjoint sets of N_p individuals on M commonly agreed upon SNPs, where the notation $[P] := \{1, 2, \ldots, P\}$. Let $N = \sum_{p \in [P]} N_p$ be the total number of individuals in the joint dataset. The data shared among all parties can be represented as a *genotype matrix* $\boldsymbol{X} \in \{0, 1, 2\}^{N \times M}$, where entry $X_{i,j} \in \{0, 1, 2\}$ is the ith individual's allele count for SNP j. Explicitly, an individual heterozygous for the variant SNP will have count 1, while an individual homozygous for the variant will have count 2, and an individual homozygous for the reference will have count 0. Similarly, the "local" genotype matrix each party holds for their own data is denoted $\boldsymbol{X}^{(p)} \in \{0, 1, 2\}^{N_p \times M}$. Together, the $\boldsymbol{X}^{(p)}$ are horizontal slices of the aggregate "global" genotype matrix $\boldsymbol{X}$. Likewise, let $\boldsymbol{Z} \in \mathbb{R}^{N \times M}$ denote the *standardized genotype matrix* in which each column, corresponding to a SNP, has been mean centered and has unit variance; and $\boldsymbol{Z}^{(p)} \in \mathbb{R}^{N_p \times M}$ similarly.

Using the standardized genotype matrix, it is possible to compute the linkage disequilibrium (LD) matrix. There are multiple definitions of LD, but the PRS algorithm we will implement uses the LD correlation matrix. [11]

Definition 1.3.1 (Linkage Disequilibrium matrix)**.** Given standardized genotype matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times M}$, the LD matrix $\boldsymbol{D} \in [-1, 1]^{M \times M}$ is defined as

$$\boldsymbol{D} := \frac{\boldsymbol{Z}^\top \boldsymbol{Z}}{N},$$

where $\boldsymbol{D}_{i,j}$ is the linkage disequilibrium between SNPs i and j. [11], [13]

I note properties of the LD matrix which will later be algorithmically relevant.

Property 1.3.1. The LD matrix is symmetric.

Property 1.3.1 owes to the fact that the LD is definitionally computed as $A^\top A$ for a real-valued matrix A, which implies symmetricness.

Property 1.3.2 (Additive distribution of LD matrix)**.** Given standardized $N \times M$ genotype matrix $\boldsymbol{Z}$ is horizontally split into $N_p \times M$ local genotype matrices $\boldsymbol{Z}^{(p)}$, the corresponding LD matrices are related as

$$\boldsymbol{D} = \sum_{p \in [P]} \frac{N_p}{N} \boldsymbol{D}^{(p)},$$

where $\boldsymbol{D}^{(p)} := \dfrac{{\boldsymbol{Z}^{(p)}}^\top \boldsymbol{Z}^{(p)}}{N_p}$ is the pth party's LD matrix computed off of their "local" data.

This property owes to the fact that the genotype matrix is horizontally split:

$$\begin{aligned}
\boldsymbol{D} &= \frac{\boldsymbol{Z}^\top \boldsymbol{Z}}{N} \\
&= \sum_{p \in [P]} \left(\frac{{\boldsymbol{Z}^{(p)}}^\top \boldsymbol{Z}^{(p)}}{N} \right) \\
&= \sum_{p \in [P]} \frac{N_p}{N} \left(\frac{{\boldsymbol{Z}^{(p)}}^\top \boldsymbol{Z}^{(p)}}{N_p} \right) \\
&= \sum_{p \in [P]} \frac{N_p}{N} \boldsymbol{D}^{(p)}.
\end{aligned}$$

As a matter of notation, we will often instead index the parties by i, and also write $\boldsymbol{D}_i$ or $\boldsymbol{Z}_i$ to refer to the ith party's matrices.

Property 1.3.3 (All-ones diagonal of LD matrix)**.** The diagonal entries of an LD matrix are all equal to 1.

This property follows from the definition of the LD matrix as a correlation matrix. It has an intuitive interpretation: any SNP will necessarily always be inherited with itself.

1.4 Cryptography background

Our work leverages two distinct cryptographic techniques: homomorphic encryption and multiparty computation. Both techniques essentially offer a means to evaluate functions over data without revealing anything about the data (other than the output, in the case of MPC). Each technique has its own strengths and limitations, and judicious use of both techniques and interconverting between them expands the range of possible computations. Even still, both techniques are expensive to use in practice, either in terms of time or communication complexity, and it is therefore best to maximize plaintext computation when possible.

1.4.1 Homomorphic encryption (HE)

A homomorphic encryption (HE) scheme is an encryption scheme with the additional property that one can evaluate circuits over the ciphertexts. [10], [15] Colloquially, HE allows a secret-bearing party to "outsource" a computation over that secret to another party, without the computing party learning anything about the secret. Although an HE scheme definitionally allows for arbitrary circuit evaluation, in practice the feasible space of functions is quite circumscribed. Specifically, the efficiency of HE schemes is bottlenecked by the multiplicative depth of the computed function, as ciphertexts in these schemes can only be used for a fixed number of multiplications before they must be refreshed through a costly operation called *bootstrapping*. [10], [15]

In our work, we use the Cheon-Kim-Kim-Song (CKKS) HE scheme. [16] In CKKS, the message space is complex numbers: specifically vectors of complex numbers $\mathbb{C}^{N/2}$, where N is a power of 2 chosen as part of the parameterization. [1] As the message space is complex numbers, ciphertexts can be thought of as containing many "slots" for separate values that can be acted upon simultaneously in parallel. [2] This property enables single instruction, multiple data (SIMD) type parallel computations, promising greater efficiency with thoughtful algorithmic design.

CKKS derives its cryptographic security from the Ring Learning with Errors (RLWE) assumption: the practical consequence of this being that encrypting a message into a ciphertext causes it to suffer a small amount of noise in its least significant bits, and so all arithmetic with CKKS is approximate. If the noise is negligible compared to the underlying value, then the computations will remain valid; this presents a practical constraint on how small message values can be, and some care must be taken to prevent HE noise from accruing.

There are three primitive operations available for ciphertexts in CKKS:

1. $\mathsf{Add} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, which adds two ciphertexts pointwise between slots

2. $\mathsf{Mult} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, which multiplies two ciphertexts pointwise between slots

3. $\mathsf{Rot} : \mathcal{C} \times \mathbb{Z} \to \mathcal{C}$, which shifts the slots of a ciphertext by mapping the ith slot to the $(i + k)$th slot, wrapping cyclically around.

To give a sense of the relative costliness of the different operations, addition is the fastest, followed by multiplication, and rotation is the slowest, equivalent to two multiplications. Bootstrapping is an order of magnitude costlier than a multiplication. It is also possible to Add and Multiply between a ciphertext and a plaintext value; plaintext-ciphertext multiplication is faster than ciphertext-ciphertext multiplication by a factor of about eight. [7], [8]

[1] There is also a real-valued version of CKKS, but our implementation is based off of the original complex-valued version.

[2] To give a sense of scale, in our parameterization of CKKS, $N = 2^{14}$: so each ciphertext has $2^{13} = 8192$ slots.

1.4.2 Secure multiparty computation (MPC)

Multiparty computation (MPC) is a technique where two or more parties can jointly evaluate a function of their inputs without learning anything about the other parties' inputs. Put symbolically, if each party i bears an input value x_i, they can collaborate using MPC to compute a function $y = f(x_1, \ldots, x_k)$ without learning about any other party's x_i value. [10], [17] Our work specifically uses *additive secret sharing* to perform MPC, meaning that at each step of the computation, each party possesses a *secret share* s_i such that the sum of the shares equals the current value. Each share is "blinded" so that it reveals no information whatsoever about the true value underlying the computation, and in fact that any subset of the shares reveal no information. In such a scheme, the secret shares are actually elements of some field, such as $\mathbb{Z}_q$, the field of integers modulo q for a prime q. As the actual underlying values of the MPC scheme are integers, real numbers are approximately encoded by scaling them up by some constant 2^f and snapping them to the nearest integer. [9]

To give a toy example for concreteness, suppose one wishes to share a secret value x with two parties, Alice and Bob. Suppose $x \in \mathbb{Z}_q$, the field of integers modulo q for a prime q. Let r be a uniformly sampled element of $\mathbb{Z}_q$. We can give Alice $x - r$ and Bob r. Clearly by construction the secret shares are additive; and they are also perfectly hiding, as both $x - r$ and r are uniformly distributed over $\mathbb{Z}_q$. This idea can be generalized to many parties by independently sampling many $r_1, \ldots, r_k$, giving Alice $x - \sum_i r_i$ and party i the value r_i. More complicated techniques must be used to implement multiplication and other functions; but it is possible to implement the following functions:

1. MPC-SqrtAndSqrtInverse(a): Returns (secret shares of) approximations of $\sqrt{a}$ and $1/\sqrt{a}$.
2. MPC-Divide(a, b): Returns an approximation of a/b.
3. MPC-IsPositive(a): Returns 1 if $a > 0$ and 0 otherwise.
4. MPC-GreaterThan(a, b): Returns 1 if $a > b$ and 0 otherwise.
5. MPC-LessThan(a, b): Returns 1 if $a < b$ and 0 otherwise.

There are also versions of the inequality functions where one argument is public, which can be more efficient. See the supplement in [9] for more information about the protocols to compute the above functions.

As a last detail, our implementation of MPC requires the presence of a "dealer" party, denoted party 0. Specifically, the dealer performs certain pre-processing computations prior to the interaction of the main, computing parties to enable more efficient multiplication. [9], [10]

Though MPC provides a means to compute functions that would be prohibitively expensive to compute with HE, MPC requires communication between parties to evaluate functions whereas HE can be done locally. There are ways of converting from CKKS ciphertexts to additive secret shares and vice-versa, allowing us to design algorithms that leverage benefits from both schemes. [7]

Chapter 2

Methods

I begin this chapter with a review of the PRS-CS algorithm and the form of its input data. Then, I discuss our primary technical contributions: general algorithmic techniques we used to address technical constraints arising from using HE/MPC, and a generic technique for secure random variate sampling. Finally, I present each of our sampling algorithms and detail the considerations underpinning their design.

2.1 Review of PRS-CS Algorithm

The PRS-CS algorithm performs a Bayesian regression using a Gibbs sampling algorithm. A **Gibbs sampler** is a type of Markov Chain Monte Carlo (MCMC) algorithm for approximating samples from a multivariate probability distribution where direct sampling from the joint distribution is intractable. Let $f(\boldsymbol{\theta})$ be a multivariate probability distribution of interest, parameterized by the vector of variables $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$. A Gibbs sampler reduces sampling the multivariate $\boldsymbol{\theta}$ to a series of univariate samplings, in which each coordinate is sampled conditioned upon the others, $\theta_i \mid \boldsymbol{\theta}_{-i}$, where $\boldsymbol{\theta}_{-i} := (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_k)$ denotes the vector $\boldsymbol{\theta}$ excluding the ith coordinate. Thus, if we would ordinarily draw n samples from $f(\boldsymbol{\theta})$, in using the Gibbs sampler we would instead, each iteration, draw a sample for each θ_i in turn, updating as we go. [18]

The Bayesian regression model for PRS-CS is:

$$\boldsymbol{y} = \boldsymbol{Z}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}), \quad p(\sigma^2) \propto \sigma^{-2}, \tag{2.1}$$

where $\boldsymbol{Z}$ is the $N \times M$ standardized genotype matrix; $\boldsymbol{y}$ is standardized phenotype vector of our training population; ε denotes the vector of independent environmental effects; and the residual variance σ^2 has been assigned a non-informative scale-invariant Jeffreys prior $p(\sigma^2)$.

The β_j are assigned a "global-local scale mixture of normals" prior:

$$\beta_j \sim \mathcal{N}\left(0, \frac{\sigma^2}{N}\phi\psi_j\right), \quad \psi_j \sim g, \tag{2.2}$$

where ψ_j is a local, SNP-specific shrinkage parameter; ϕ is a "global shrinkage parameter"; and g is an absolutely continuous mixing density function. For PRS-CS, the particular prior on ψ_j is the independent gamma-gamma prior:

$$\psi_j \sim \text{Gamma}\left(a, \delta_j\right), \quad \delta_j \sim \text{Gamma}\left(b, 1\right), \tag{2.3}$$

where a, b are hyperparameters and $\text{Gamma}\left(k, \theta\right)$ denotes the gamma distribution with shape parameter k and scale parameter θ. PRS-CS uses the hyperparameter values $a = 1$ and $b = 1/2$, which is called the Strawderman-Berger prior or the quasi-Cauchy prior, as it was found to "work well across a range of simulated and real genetic architectures." [11]

PRS-CS has two versions: base PRS-CS, in which the global shrinkage parameter ϕ is fixed; and PRS-CS-auto, where ϕ is itself estimated. Our work investigates base PRS-CS with the recommended value $\phi = 0.01$, though an extension to PRS-CS-auto is a natural realm of future work.

2.1.1 The PRS-CS Gibbs sampler

The Gibbs sampler for PRS-CS samples the four parameters $\beta, \sigma^2, \delta, \psi$ in order as follows: [11]

Sample β:

$$\left[\beta \mid \sigma^2, \mathbf{\Psi}, \hat{\beta}, \boldsymbol{D}\right] \sim \text{MVN}\left(\mu, \mathbf{\Sigma}\right), \quad \mu = \frac{N}{\sigma^2}\mathbf{\Sigma}\hat{\beta}, \quad \mathbf{\Sigma} = \frac{\sigma^2}{N}\left(\boldsymbol{D} + \mathbf{\Psi^{-1}}\right)^{-1} \tag{2.4}$$

Sample σ^2: [1]

$$\begin{gathered}\left[\sigma^2 \mid \beta, \mathbf{\Psi}, \hat{\beta}, \boldsymbol{D}\right] \sim \text{invGamma}\left(\frac{N+M}{2}, \frac{N}{2}\varsigma\right), \\ \varsigma = \max\left\{1 - 2\beta^\top\hat{\beta} + \beta^\top(\boldsymbol{D} + \mathbf{\Psi^{-1}})\beta, \quad \beta^\top\mathbf{\Psi^{-1}}\beta\right\},\end{gathered} \tag{2.5}$$

Sample δ_j:

$$\left[\delta_j \mid \psi_j\right] \sim \text{Gamma}\left(a + b, \frac{1}{\psi_j + \phi}\right) \tag{2.6}$$

Sample ψ_j: [2]

$$\begin{gathered}\left[\psi_j^{\text{temp}} \mid \beta_j, \sigma^2, \delta_j\right] \sim \text{giG}\left(a - \frac{1}{2}; 2\delta_j, \frac{N\beta_j^2}{\sigma^2}\right), \\ \psi_j = \min\{1, \psi_j^{\text{temp}}\},\end{gathered} \tag{2.7}$$

where $\text{MVN}\left(\mu, \mathbf{\Sigma}\right)$ denotes the multivariate normal with mean μ and covariance $\mathbf{\Sigma}$; $\text{Gamma}\left(\alpha, \beta\right)$ and $\text{invGamma}\left(\alpha, \beta\right)$ denote the gamma and inverse gamma distributions, respectively, with shape parameter α and scale parameter β; and $\text{giG}\left(p;\ \rho, \chi\right)$ denotes the generalized inverse Gaussian distribution.

We reproduce the pseudocode for the PRS-CS Gibbs sampler in Algorithm 1. We use Ge et al.'s recommended default MCMC parameters: 1000 MCMC iterations with 500 burn-in iterations and a thinning factor of 5.

[1]While omitted from the supplement to PRS-CS, the max between possible values of ς is implemented in the codebase, and necessary to ensure that $\varsigma > 0$ for all inputs.

[2]PRS-CS as an algorithm restricts $\phi^{-1}\psi_j^{-1} \geq \rho$ for $\rho = 1$. The PRS-CS codebase satisfies this restriction by clamping ψ_j above at 1, though this is a stronger bound than is necessary for small ϕ. Nevertheless, we implement this bound to match the original implementation.

Algorithm 1 PRS-CS.

Input: The number of MCMC iterations, the number of burn-in iterations, and the thinning rate; the GWAS effect size estimates $\hat{\beta}$ and each party's local LD matrix $\boldsymbol{D}_i$; population count N and SNP count M; PRS-CS hyperparameters a, b, ϕ.

Output: PRS effect sizes β.

```
1: Initialize $\sigma^2 = 1$; and $\beta_j = 0$ and $\psi_j = 1$ for all $j \in [M]$.
2: for $t \in \{1, 2, \ldots, mcmc\_iters\}$ do
3:     Sample $\beta$.
4:     Sample $\sigma^2$.
5:     Sample $\delta_j$ for all $j \in [M]$.
6:     Sample $\psi_j$ for all $j \in [M]$.
7:     Clamp all $\psi_j$ to $\min\{\psi_j, 1\}$.
8:     if $t > burn\_in\_iters$ and $t$ is divisible by $thinning\_rate$ then
9:         Keep $\beta^{(t)}$.
10:    end if
11: end for
12: return the average of the kept betas $\bar{\beta} = \sum_{\text{kept } t} \beta^{(t)}$.
```

2.1.2 Structure of the input data to PRS-CS

PRS-CS uses two types of input data: GWAS-derived effect sizes $\hat{\beta}$, and the linkage disequilibrium matrix $\boldsymbol{D}$. Notably, both of these inputs scale only in the number of SNPs M and not in the number of individuals N, encouraging larger collaborations at no added computational burden. Though we assume that each party has its own genotype data $\boldsymbol{X}_i$, we assume that $\hat{\beta}$ is public, as in practice GWAS effect sizes are published in the literature.

Without additional assumptions, it would be intractable to perform PRS-CS, as sampling β would be infeasible even in plaintext. Accordingly, Ge et al. make the additional assumption that the LD matrix can be partitioned into blocks that are approximately independent from each other: LD effects between SNPs within the same block are non-negligible, but LD effects between SNPs of different blocks are negligible.[11] By making this assumption, $\boldsymbol{D}$ is understood to have block diagonal structure, and can be partitioned into separate LD blocks $\boldsymbol{D}_\ell$. Sampling of β can therefore be done only at the block level, with each block's β_ℓ drawn independently of the others.

The "approximately independent linkage disequilibrium blocks" can be computed using an algorithm by Berisa and Pickrell. [19] In our work, we take the approximately independent cutoffs published by Berisa and Pickrell, though future work could also incorporate the cutoff finding algorithm into the secure federated PRS pipeline. In practice, these blocks have dimension of a couple hundred up to a thousand, though the target block size is a tunable parameter of the LD block finding algorithm.

Though the original intent of recognizing the approximate block structure of the LD matrix was to improve the computational tractability of β sampling, for our purposes it is actually desirable to group LD blocks together into larger units. Recall that CKKS ciphertexts encode vectors of multiple values, in our case $2^{13} = 8192$. We are therefore encouraged to batch computations for optimal efficiency, but this appears to be in tension with the efficiency of β sampling. However, as will be discussed in greater detail in Section 2.6, for the major bottlenecks to β sampling efficiency—matrix multiplication and solving a system of linear equations with an iterative algorithm—the runtime cost of both of these operations will either grow slowly or not at all in the underlying dimension of a ciphertext. Moreover, as the ψ_j, δ_j are sampled independently, it is desirable to take advantage of parallel computation to its fullest extent by packing ciphertexts as fully as possible. It is therefore on balance advantageous to group blocks together. In our work, we arbitrarily group LD blocks together using a greedy algorithm, though more sophisticated bin packing algorithms could potentially improve the runtime of the downstream β sampling task; see the Appendix for more detail.

2.2 Designing sampling algorithms within the cryptographic framework

In this section, I detail some of the more general techniques we devised to overcome technical constraints of implementing algorithms using HE/MPC. I will then discuss a very general solution for secure random variate sampling—sampling from a standard distribution and rescaling—and why it was insufficient for implementing all algorithms in our application.

2.2.1 Overcoming technical constraints in secure computing

HE and MPC each pose constraints to the computational model available, creating new challenges that must be navigated in refactoring an algorithm. In general, all HE and MPC operations are significantly more computationally intensive than their plaintext counterparts; and MPC operations require communication between parties and will incur latency and network costs accordingly. Commonplace operations that would be cheap in plaintext are made significantly slower under HE and MPC, and familiar algorithmic approaches become profligate or unworkable.

Deftly navigating the constraints posed by the cryptographic framework entails unfamiliar algorithmic rewrites. For example: in plaintext, computing the quantity 4λ using addition of λ to itself instead of by multiplying by 4 would be non-idiomatic and unusual, but inconsequentially so. But under HE, it is better to compute 4λ with addition, as it is faster than multiplication and does not consume any ciphertext levels. Many of the implemented algorithmic rewrites are of this type; we will therefore present many algorithms in two forms: one form that is most readily understood, and one form that includes rewrites that obscure the underlying algorithm yet better respect the constraints of our computational model.

Though the sampling algorithms exhibited herein are specific to our application context, we believe that the reasoning and thought process used in developing these algorithms may be of aid for future work. If anything, the way in which these sampling algorithms are specific to our problem—which variables are public and which are sensitive; how data is distributed among parties; expectations for where quantities might become too large or too small and overflow magnitude constraints—may itself signify a lesson about this problem context. The best sampling algorithms in context may require bespoke algorithms in order to exploit any available properties.

Addressing magnitude constraints through averaging

While all computers face limits on the range of representable values, these restraints are more palpable in the context of both homomorphic encryption and multiparty computation. If a quantity that is too large is ever computed, the resulting ciphertexts/secret shares will be ill-formed and unusable. As a result, we must sometimes rewrite algorithms to avoid the computation of large quantities by first multiplying through by a fraction before summing multiple terms to take an average of the terms, which can later be rescaled to the sum. It is also important to design the algorithms mindful of the fact that the input size N, M may vary: any algorithmic choices to confine the anticipated magnitude of variables within a particular range must be robust to various input sizes. In our context, the main operation that might exceed the representable range is the inner product, for which we take an average over the coordinates.

Addressing precision constraints through upscaling

In HE, the smallest representable quantities are dictated by the presence of HE noise, the noise added to all messages to invoke the (Ring) Learning with Errors assumption that powers the security of the scheme. [16] In MPC, precision is constrained by the size of the finite field used in the secret sharing scheme. [9] Many of the techniques used ordinarily to prevent floating point roundoff error are also relevant in this context: noise only affects the low order bits of the representations of numbers, so it is better ceteris paribus to compute larger quantities whenever possible, and avoid representing small numbers. The issue of precision is particularly pointed in our problem context given that many of the PRS-CS parameters, such as $\beta, \hat{\beta}$, and ψ_j, reside in the $\approx 10^{-5}$ to 10^{-2} range. HE noise under our CKKS parameter settings would ordinarily be about the size of the SNP effect sizes and potentially affect results. This is normally arithmetically fine, as the quantity is still approximately correct, but many of the MPC operations assume positive inputs, such as MPC-Sqrt and the denominator to MPC-Divide, and will produce ill-formed outputs if this precondition is violated. We must therefore take care whenever converting a ciphertext with small values to secret shares and using those values in an MPC operation.

We address the issues of precision and HE noise by upscaling each of the small parameters that arise in our problem context: redesigning the algorithms to sample from different distributions than the target, e.g. $k_\beta \beta$ instead of β for some known, tunable constant k_β. Upscaling can also provide benefits for improving the numerical stability of MPC operations by feeding them larger inputs. There is a tension between resolving issues of precision and the aforementioned issue of magnitude, as addressing precision encourages representing large quantities, but addressing magnitude discourages representing large quantities. Recalling the issue of large inner products, averaging by multiplying through a small constant might cause the individual terms to vanish below HE noise; and taking inner products over values with upscaling might exceed the upper end of representable numbers. We use our expectations of the approximate size of the quantities to justify our choices of the scaling constants k, and also to carefully design our algorithms so that neither magnitude nor precision constraints are flouted. In Table 2.1, we describe the approximate expected ranges of each of the PRS-CS parameters and inputs, which helps motivate the choices of rescaling constants used. Note

Parameter	Approximate range
β_j	$[10^{-5}, 10^{-2}]$
σ^2	$[.5, 3]$
δ_j	$[10^{-1}, 10^{3}]$
ψ_j	$[10^{-4}, 1]$
ψ_j^{-1}	$[1, 10^{4}]$
$\hat{\beta}_j$	$[10^{-5}, 10^{-1}]$
$\boldsymbol{D}_{i,i}$	$[10^{-5}, 1]$

Table 2.1: Approximate expected ranges of PRS-CS parameters and inputs

that the smallest quantity that can be represented in CKKS is about 10^{-5} or so, artificially making that value the lower bound.

Ultimately, we rescale β_j and $\hat{\beta}$ by a constant k_β; and rescale ψ_j by a constant k_ψ. In our implementations, we use the values $k_\beta = k_\psi = 10^3$.

2.2.2 The sample-and-rescale approach to secure random variate generation

The simplest idea for random variate sampling under encryption is also one that works: to sample from a standard form, which can be done in plaintext, and cryptographically rescale to the target distribution. For example, the gamma distribution $\text{Gamma}(k, \theta)$ with shape parameter k and scale parameter θ has the following property:

Property 2.2.1 (Gamma distribution scaling property)**.** If $X \sim \text{Gamma}(k, \theta)$, then $cX \sim \text{Gamma}(k, c\theta)$.

To sample from a distribution $\text{Gamma}(k, \theta)$ with public k but private θ, one can use Property 2.2.1, sample in plaintext from $\text{Gamma}(\alpha, 1)$, and use homomorphic multiplication to rescale the sample to one from $\text{Gamma}(\alpha, \theta)$. The usefulness of the sample-and-rescale technique can be further enhanced by considering relationships between probability distributions, which both expands the scope of the distributions accessible with this technique and may also suggest alternative, more computationally efficient ways of sampling from a particular distribution. Consider the following scenario: one must sample a variable $X \sim \text{invGamma}(a, b)$, and both X and $1/X$ will be used in some downstream computation. There is a relationship between the gamma and inverse-gamma distributions:

Property 2.2.2 (Relationship between Gamma and Inverse Gamma distributions)**.** If $X \sim \text{Gamma}(k, \theta)$, then $1/X \sim \text{invGamma}(k, 1/\theta)$.

So either X or $1/X$ can be sampled, and the one quantity derived from the other. If this were plaintext sampling, one would be largely indifferent as to whether X were sampled from the gamma or inverse gamma distribution, as both are readily available. In the cryptographic context, they are not equivalent. If a is public and b is private, it is preferable to sample from $\text{invGamma}(a, 1)$ and rescale to $\text{invGamma}(a, b)$ using the inverse gamma rescaling property:

Property 2.2.3 (Gamma distribution scaling property)**.** If $X \sim \text{invGamma}(\alpha, \beta)$, then $cX \sim \text{Gamma}(\alpha, c\beta)$.

As this rescaling can be done with a cryptographic multiplication with b. In contrast, to sample from the equivalent gamma distribution would require sampling from $\text{Gamma}(a, 1)$ and a cryptographic multiplication with $1/b$ instead: computing this quantity would require an MPC Divide, a relatively expensive operation. Awareness of the relationships between probability distributions may both present a broader range of accessible distributions and suggest algorithmic optimizations.

Limitations to the sample-and-rescale approach

While obvious and widely useful, the sample-and-rescale approach has limitations. Notably, this approach requires that one has access both to: (i) a source of samples from the standard form of the distribution and (ii) the relevant transformation factors. The first criterion might not be satisfied if the distribution has no readily available plaintext sampling function. The second criterion might not be satisfied if the quantities necessary for the transformation cannot be easily computed, or if the transformation necessary is too computationally involved. Both criteria did not hold for some distributions in our implementation of PRS-CS. For ψ_j sampling, neither the generalized inverse Gaussian, nor the inverse Gaussian that it reduces to in our case, had readily available plaintext sampling implementations in Golang's scientific computing libraries. For β sampling, standard MVN samples can be easily obtained, but the rescaling factor is prohibitively expensive to compute, owing to how the covariance matrix is private and distributed among parties. The tacks taken for those distributions will be detailed in their sections; but the basic sample-and-rescale approach did work for sampling σ^2 and δ_j.

Because the HE/MPC framework encourages non-idiomatic computational rewrites for the sake of efficiency, I will present many algorithms once in its essential and most easily understood form; then a second time in its revised form, so that the differences can be highlighted. Even still, we will abstract away from particulars of whether data is represented as ciphertexts in HE or secret shares in MPC, and omit implied conversions between HE and MPC, bootstrapping operations, and communication of variables between parties. Generally speaking, variables should be assumed to be single HE ciphertexts unless otherwise indicated: said to be plaintext, or involved in an MPC operation.

2.3 Algorithm for secure δ_j sampling

For clarity, we reproduce the probability distribution in Equation 2.6 that the δ_j are sampled from below (and will do so for the other distributions as well.) The δ_j are sampled independently for each SNP $j \in [M]$ as

$$\delta_j \sim \text{Gamma}\left(a + b, \frac{1}{\psi_j + \phi}\right).$$

The parameter δ_j is the most straightforward to sample in our context. The hyperparameters a, b are fixed and public, as they determine what priors are being used in the Bayesian regression. We can therefore implement δ_j sampling directly by using the gamma rescale property 2.2.1:

Algorithm 2 Sampling for δ_j.

```
    **Input:** Hyperparameters $a, b, \phi$; parameters $\psi_j$; source of gamma samples.
1: For each $j \in [M]$, sample $X_j \sim \text{Gamma}(a+b, 1)$.
2: Compute the *rate* $\rho_j = \psi_j + \phi$.
3: Scale the samples as $\delta_j = X_j/\rho_j$.
4: **return** the samples.
```

We reproduce the algorithm in its revised form as Algorithm 3. The major change is accounting for the fact we upscale ψ_j to $k_\psi \psi_j$ and must account for this rescaling in computing the rate.

Algorithm 3 Secure sampling for δ_j.

```
    **Input:** Hyperparameters $a, b, \phi$; public constant $k_\psi$; parameters $\psi_j$; source of gamma samples.
1: For each $j \in [M]$, sample $X_j \sim \text{Gamma}(a+b, 1)$ in plaintext.
2: Upscale the samples to $X'_j = k_\psi X_j$.
3: Compute the upscaled rate as $\rho'_j = k_\psi \psi_j + k_\psi \phi$.
4: Compute the samples as $\delta_j = X'_j/\rho'_j$ using MPC-Divide.
5: **return** the samples.
```

2.4 Algorithm for secure σ^2 sampling

The parameter σ^2 is sampled as

$$\sigma^2 \sim \text{invGamma}\left(\frac{N+M}{2}, \frac{N}{2}\varsigma\right), \quad \varsigma = \max\left\{1 - 2\beta^\top \hat{\beta} + \beta^\top(\boldsymbol{D} + \boldsymbol{\Psi}^{-1})\beta, \ \ \beta^\top \boldsymbol{\Psi}^{-1}\beta\right\},$$

where $\boldsymbol{\Psi}^{-1}$ denotes the diagonal matrix whose jth diagonal entry is ψ_j^{-1}.

Sampling σ^2 is relatively straightforward via the inverse gamma rescaling property 2.2.3, but there are two problems: implementing the max function, and ensuring that no quantities involved become unrepresentably large.

Implementing the max can be done with the observation that we can write the left possibility for ς with a separate term that is the right possibility:

$$1 - 2\beta^\top \hat{\beta} + \beta^\top(\boldsymbol{D} + \boldsymbol{\Psi}^{-1})\beta = (1 - 2\beta^\top \hat{\beta} + \beta^\top \boldsymbol{D}\beta) + (\beta^\top \boldsymbol{\Psi}^{-1}\beta).$$

For clarity, let us refer to the left expression in the max as ς_L, the right expression as ς_R, and the first parenthetical term above as ς_{L-R}: the left possibility minus the right possibility. If the quantity $\varsigma_{L-R} = 1 - 2\beta^\top \hat{\beta} + \beta^\top \boldsymbol{D}\beta$ is positive, then $\varsigma_L = \varsigma_{L-R} + \varsigma_R > \varsigma_R$. We can compute ς_{L-R} in HE, use MPC to check if it is positive, and then compute the correct scale accordingly.

The second concern owes to the fact that σ^2 is a single scalar regardless of input size, whereas the other parameters β, δ_j, ψ_j are sampled locally: β is sampled independently between LD blocks; and δ_j, ψ_j are sampled independently per SNP. In contrast, the private scale parameter for σ^2 involves terms that can grow proportionately to the number of SNPs in the input data: $\beta^\top \hat{\beta}$, $\beta^\top \boldsymbol{D} \beta$, and $\beta^\top \boldsymbol{\Psi^{-1}} \beta$. Based on the approximate anticipated ranges of these quantities (see 2.1), it is primarily a concern that $\beta^\top \boldsymbol{\Psi^{-1}} \beta$ could grow to a large size, whereas it is anticipated that $\beta^\top \hat{\beta}$ and $\beta^\top \boldsymbol{D} \beta$ may roughly be of the same size. The quantity $\beta^\top \boldsymbol{D} \beta$ can be computed in CKKS with minimal communication overhead owing to the fact that each party's local LD matrix sums to the global LD matrix via 1.3.2. More information on the particular matrix multiplication algorithm from Jiang et al. [20] is in the Appendix.

Algorithm 4 Secure matrix multiplications with $\boldsymbol{D}$.

```
    Input: Ciphertext vector $x$, each party's local $\boldsymbol{D}_i$.
    Output: Ciphertext vector $\boldsymbol{D}x$.
1: Each party locally computes $\boldsymbol{D}_i x$ using their plaintext $\boldsymbol{D}_i$.
2: The parties aggregate $\sum_{i \in [P]} \frac{N_i}{N} \boldsymbol{D}_i x = \boldsymbol{D} x$.
3: Return $\boldsymbol{D}x$.
```

The only last thing to remark upon is that we must account for the upscaling of β and $\hat{\beta}$. We present the revised algorithm as Algorithm 5.

Algorithm 5 Secure sampling for σ^2.

```
    Input: Public constants $N, M, k_\beta$; public GWAS statistics $\hat{\beta}$; parameters $\boldsymbol{\Psi^{-1}}, \beta$; access to inverse gamma samples.
1: Compute $\varsigma_R = \frac{1}{k_\beta^2}(k_\beta \beta^\top)\boldsymbol{\Psi^{-1}}(k_\beta \beta)$.
2: Compute $\varsigma_{L-R} = \frac{1}{k_\beta^2}\left[k_\beta^2 - 2(k_\beta \beta^\top)(k_\beta \hat{\beta}) + (k_\beta \beta^\top)\boldsymbol{D}(k_\beta \beta)\right]$.
3: Compute $\mathbb{1}\{\varsigma_{L-R} > 0\} \in \{0, 1\}$ using MPC-IsPositive.
4: Set $\varsigma = \varsigma_R + \mathbb{1}\{\varsigma_{L-R} > 0\} \cdot \varsigma_{L-R}$.
5: Sample $X \sim \text{invGamma}\left(\frac{N+M}{2}, \frac{N}{2}\right)$ in plaintext.
6: Return $\varsigma X$.
```

2.5 Algorithm for secure ψ_j sampling

The ψ_j are sampled independently for each SNP $j \in [M]$ as

$$\psi_j \sim \text{giG}\left(a - \frac{1}{2};\, 2\delta_j,\, \frac{N\beta_j^2}{\sigma^2}\right),$$

where $\text{giG}(p;\, \rho, \chi)$ denotes the generalized inverse Gaussian (giG) distribution, which has the probability density function

$$f_{\text{giG}}(x; p, \rho, \chi) = \frac{(\rho/\chi)^{p/2}}{2K_p(\sqrt{\rho\chi})} x^{p-1} e^{-(\rho x + \chi/x)/2}, \quad x, \rho, \chi > 0, \quad p \in \mathbb{R},$$

where K_p denotes the modified Bessel function of the second kind. [21] As its name suggests, the generalized inverse Gaussian generalizes several probability distributions and reduces to other distributions under particular values of p. Under the particular value $p = -1/2$, the giG reduces to the inverse Gaussian (invGauss) distribution:

Property 2.5.1 (Relationship between giG and invGauss)**.** If $X \sim \text{giG}(p = -1/2;\, a, b)$, then $X \sim \text{invGauss}\left(\mu = \sqrt{b/a}, \lambda = b\right)$. [22]

We also exploit a second property of the giG, which is the antisymmetry in p of its other two parameters:

Property 2.5.2 (Antisymmetry of the giG)**.** If $X \sim \text{giG}(p;\, a, b)$, then $1/X \sim \text{giG}(-p;\, b, a)$. [23]

Recall that PRS-CS uses the Strawderman-Berger prior, also known as the quasi-Cauchy prior, which corresponds to $a = 1$ and $b = 1/2$. [11] Hence $p = 1 - \frac{1}{2} = \frac{1}{2}$, and we can invoke both Property 2.5.2 and Property 2.5.1 together to reduce sampling from $\text{giG}(p = 1/2;\, \cdot, \cdot)$ to sampling from the inverse Gaussian distribution. We present this as Algorithm 6.

Algorithm 6 Sampling from $\text{giG}\left(p = \frac{1}{2};\, a, b\right)$.

```
    Input: giG parameters $a, b$; access to inverse Gaussian samples.
1:  Sample $x \sim \text{invGauss}\left(\mu = \sqrt{\frac{a}{b}}, \lambda = a\right)$.
2:  return $1/x$.
```

There exists an efficient algorithm for sampling from the inverse Gaussian distribution by transforming samples from the standard normal distribution $\mathcal{N}(0, 1)$ and the standard uniform distribution $U(0, 1)$, which we reproduce as Algorithm 7. [24]

The primary challenge in implementing Algorithm 7 is in parsimoniously computing all of the quantities involved, reusing as many intermediates as possible and eliminating multiplications and especially divisions. The most significant revisions in the rewrite are eliminating the division in line 5 for efficiency and rewriting the return to avoid use of control flow logic. The MPC comparison functions (less than, greater than, is positive, etc.) return secret shares of an indicator $x \in \{0, 1\}$ to denote true/false; we can arithmetize an if-else statement to the below expression, which only requires a single multiplication:

$$\mathsf{BinaryInterleave}(x; A, B) := Ax + B(1 - x) = B + (A - B)x, \tag{2.8}$$

so that A is the value returned when x is true, and B when x is false. This arithmetization allows for the interleaving of entire ciphertexts or vectors of secret shares, and so an entire group of samples can be processed in parallel. We aggregate the changes to the inverse Gaussian algorithm in Algorithm 8.

Algorithm 7 Sampling from invGauss(μ, λ). [24]

Input: Inverse Gaussian parameters μ, λ; access to standard normal and standard uniform samples.

```
1: Sample $\nu \sim \mathcal{N}(0,1)$.
2: Set $y = \nu^2$.
3: Set $x = \mu + \frac{\mu^2 y}{2\lambda} - \frac{\mu}{2\lambda}\sqrt{4\mu\lambda y + \mu^2 y^2}$.
4: Sample $z \sim U(0,1)$.
5: if $z \leq \frac{\mu}{\mu + x}$ then
6:     return $x$.
7: else
8:     return $\frac{\mu^2}{x}$.
9: end if
```

Algorithm 8 Secure sampling from invGauss(μ, λ). [24]

```
1: Sample $\nu \sim \mathcal{N}(0,1)$.
2: Set $y = \nu^2$.
3: Compute $\mu y$, $2\lambda$, and $4\lambda$.
4: Compute $\sqrt{\mu y}$ and $\sqrt{4\lambda + \mu y}$ using MPC-Sqrt.
5: Compute $\frac{1}{2\lambda}\left(\mu y - \sqrt{\mu y(4\lambda + \mu y)}\right)$ using MPC-Divide.
6: Compute $x' = 1 + \frac{1}{2\lambda}\left(\mu y - \sqrt{\mu y(4\lambda + \mu y)}\right)$, noting that $x' = x/\mu$.
7: Sample $z \sim U(0,1)$.
8: Compute $I = \mathbb{1}\{z(1 + x') \leq 1\} \in \{0,1\}$ using MPC-LessThan.
9: Compute $\mu x' = x$ and $\mu/x' = \mu^2/x$, the two return values.
10: return BinaryInterleave$(I, \mu x', \mu/x')$.
```

Putting these components together, I present the ψ_j sampling algorithm as Algorithm 9, including the scaling of ψ_j by k_ψ. Clamping of ψ_j's maximum value at 1 is achieved by clamping ψ_j^{-1}'s *minimum* value at 1. The last algorithmic trick is to realize that $\mathbf{\Psi^{-1}}$ appears in downstream calculations and to therefore keep it, treating Algorithm 8 as a ψ_j^{-1} sampling subroutine.

Algorithm 9 Secure sampling for $k_\psi \psi_j,\ \psi_j^{-1}$.

```
Input: Public constants $k_\beta, k_\psi, N$; parameters $\sigma^2, \delta_j, \beta_j$; access to standard normal and standard uniform samples.
1: Compute $\lambda = 2\delta_j$.
2: Compute $|k_\beta \beta_j| = \mathsf{BinaryInterleave}(k_\beta\beta_j > 0; k_\beta\beta_j, -k_\beta\beta_j)$ using MPC-IsPositive.
3: Compute $\sqrt{2\delta_j}$ and $\frac{k_\beta \sigma}{\sqrt{N}}$ using MPC-Sqrt.
4: Compute $\frac{\sigma}{|\beta_j|\sqrt{N}}$ from the above using MPC-Divide.
5: Compute $\mu = \frac{\sigma\sqrt{2\delta_j}}{|\beta_j|\sqrt{N}} = \sqrt{\frac{2\delta_j\sigma^2}{N\beta_j^2}}$.
6: Sample $\psi_{j,\text{temp}}^{-1} \sim \text{invGauss}(\mu, \lambda)$.
7: Set $\psi_j^{-1} = \mathsf{BinaryInterleave}(\psi_{j,\text{temp}}^{-1} > 1; \psi_{j,\text{temp}}^{-1}, 1)$ using MPC-GreaterThan.
8: Compute $\frac{k_\psi}{\psi_j^{-1}} = k_\psi\psi_j$ using MPC-Divide.
9: return $k_\psi\psi_j,\ \psi_j^{-1}$.
```

2.6 Algorithm for secure $\boldsymbol{\beta}$ sampling

The SNP effect sizes β are sampled as

$$\beta \sim \text{MVN}(\mu, \mathbf{\Sigma}),\ \mu = \frac{N}{\sigma^2}\mathbf{\Sigma}\hat{\beta},\ \mathbf{\Sigma} = \frac{\sigma^2}{N}\left(\boldsymbol{D} + \mathbf{\Psi^{-1}}\right)^{-1}.$$

The multivariate normal (MVN) distribution posed the most algorithmic challenges in this work, and accordingly this section will have several subsections to discuss some of the particular assumptions and algorithmic insights needed to make sampling tractable. Even in plaintext, the task of sampling from a high-dimensional MVN can be computationally intensive. For example, consider the naive approach of sampling from the standard MVN and rescaling to the target distribution, which relies on Property2.6.1 of the MVN.

Property 2.6.1 (Affine transformation of the MVN)**.** Let $X \sim \text{MVN}(\mu, \mathbf{\Sigma})$, where X is an n-dimensional vector. Let $Y = c + \boldsymbol{B}X$ for m-dimensional vector c and $m \times n$ matrix $\boldsymbol{B}$. Then $Y \sim \text{MVN}\left(c + \boldsymbol{B}\mu, \boldsymbol{B}\mathbf{\Sigma}\boldsymbol{B}^\top\right)$.

Accordingly, to sample from an MVN with target covariance matrix $\mathbf{\Sigma}$, one must possess the Cholesky factor of $\mathbf{\Sigma}$.

Definition 2.6.1 (Cholesky decomposition)**.** The Cholesky decomposition of a symmetric positive semidefinite (PSD) matrix $\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$ is a decomposition of the form

$$\mathbf{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top,$$

where $\boldsymbol{L} \in \mathbb{R}^{d\times d}$ is a lower triangular matrix with positive diagonal entries, called the Cholesky factor.

Computing the Cholesky factor of a $d \times d$ matrix requires $\Theta(d^3)$ operations and $\Theta(d^2)$ space. [25] Concretely, when using 64-bit floating point numbers, generating a sample for a $d = 10^5$ matrix in this way (the "Cholesky sampler") would require 40 gigabytes of memory and roughly 10^{14} floating point operations (flops). [25] If $\mathbf{\Sigma}$ possesses any additional structure, it may be possible to use a different sampler and obtain a better runtime. It is worth discussing the particular structure of our $\mathbf{\Sigma}$ to understand what constraints must be worked with in developing a sampling algorithm.

2.6.1 Structure of the covariance matrix

Given the size of PRS studies, where the number of SNPs under study M can be on the order of tens or hundreds of thousands, even a runtime of $O(M^2)$ could be substantial. Indeed, even PRS-CS in plaintext would be computationally infeasible. As mentioned earlier in Section 2.1.2, Ge et al. make the additional assumption that the LD matrix has approximate block structure so that β sampling can be instead performed over much smaller LD blocks.[11]

Recall that the linkage disequilibrium (LD) matrix $\boldsymbol{D} \in \mathbb{R}^{M\times M}$ is split up among parties as $\boldsymbol{D} = \sum_{i\in[P]} \frac{N_i}{N}\boldsymbol{D}_i$, where $\boldsymbol{D}_i$ is the LD matrix computed off of party i's local data. As each local LD matrix share $\boldsymbol{D}_i$ is quite large, even when recognized as being a block matrix, ideally we would maximize plaintext computations with the LD matrix. In contrast, $\mathbf{\Psi^{-1}}$ is sensitive.

Taken altogether, computing $\mathbf{\Sigma}$, let alone its Cholesky factor, is intractable. Too much communication would be required for the parties to securely compute the matrix inverse. Moreover, because $\mathbf{\Psi^{-1}}$ is updated each MCMC iteration and constitutes a full-rank update to the matrix $\boldsymbol{D} + \mathbf{\Psi^{-1}}$, the $\Theta(d^3)$ matrix inverse would need to be recomputed every iteration. In contrast, the inverse of $\mathbf{\Sigma}$, which itself is just a sum of matrices, would be easier to work with. The inverse of a covariance matrix is called the *precision matrix* and is denoted $\boldsymbol{Q}$. For our MVN of interest, the precision matrix is

$$\boldsymbol{Q} = \mathbf{\Sigma}^{-1} = \frac{N}{\sigma^2}\left(\boldsymbol{D} + \mathbf{\Psi^{-1}}\right).$$

Fortunately, some of the sampling algorithms present in Vono et al. are well-suited for a situation where the precision matrix is more readily available than the covariance matrix. We will implement a version of the "perturbation-optimization sampler," which we reproduce as Algorithm 10. [25]

2.6.2 The MVN sampling algorithm

Algorithm 10 comports well with our problem context, and each of its three steps is feasible. First, it is relatively easy to draw samples from MVN $(\mathbf{0}, \boldsymbol{Q})$ using the sum-of-normals property:

Algorithm 10 Perturbation-optimization sampling from MVN $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

```
Input: The precision matrix $\boldsymbol{Q} = \boldsymbol{\Sigma}^{-1}$ and transformed mean $\boldsymbol{Q}\mu$; access to samples from MVN $(\mathbf{0}, \boldsymbol{Q})$.
1: Draw $\boldsymbol{z} \sim$ MVN $(\mathbf{0}, \boldsymbol{Q})$.
2: Set $\boldsymbol{\eta} = \boldsymbol{Q}\boldsymbol{\mu} + \boldsymbol{z}$.
3: Solve $\boldsymbol{Q}\boldsymbol{\theta} = \boldsymbol{\eta}$ with respect to $\boldsymbol{\theta}$.
4: return $\boldsymbol{\theta}$.
```

Property 2.6.2 (Sum of normal random variables)**.** If $X \sim \text{MVN}(\mu_1, \boldsymbol{\Sigma}_1)$ and $Y \sim \text{MVN}(\mu_2, \boldsymbol{\Sigma}_2)$, then $X + Y \sim \text{MVN}(\mu_1 + \mu_2, \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)$.

Each party can draw a sample from MVN $\left(\mathbf{0}, \frac{N_i}{N}\boldsymbol{D}_i\right)$ in plaintext and encrypt it; and some particular party can draw a sample from MVN $(\mathbf{0}, \boldsymbol{\Psi}^{-1})$ by rescaling a sample from the standard MVN. More concretely, we can implement step 1 of Algorithm 10 as Algorithm 11. Note that the LD matrix is a fixed input, so it is feasible to compute the Cholesky factor of each LD block once and use it for all samples from MVN $\left(\mathbf{0}, \frac{N_i}{N}\boldsymbol{D}_i\right)$.

Algorithm 11 Secure implementation of step 1 of Algorithm 10.

```
Input: Public constants $N_i, N$; parameters $\boldsymbol{\Psi}^{-1}, \sigma^2$; access to samples from MVN $(\mathbf{0}, \boldsymbol{D}_i)$ and MVN $(\mathbf{0}, \boldsymbol{I})$.
1: Each party $i$ samples a value $z_i \sim$ MVN $\left(\mathbf{0}, \frac{N_i}{N}\boldsymbol{D}_i\right)$ (e.g. with a plaintext Cholesky sampler).
2: Each party encrypts their $z_i$, and the parties aggregate $z^+ = \sum_{i\in[P]} z_i$.
3: Compute $\boldsymbol{\Psi}^{-1/2}$ using MPC-Sqrt.
4: The hub party samples $y \sim$ MVN $(\mathbf{0}, \boldsymbol{I})$ in plaintext and computes $z^* = \boldsymbol{\Psi}^{-1/2}y$.
5: Return $z = \frac{\sqrt{N}}{\sigma}(z^+ + z^*)$.
```

Next, step 2 of Algorithm 10 obviates the apparent need to compute the $\boldsymbol{\Sigma}$ that appears in μ:

$$\boldsymbol{Q}\mu = \boldsymbol{Q}\left(\frac{N}{\sigma^2}\boldsymbol{\Sigma}\hat{\beta}\right) = \frac{N}{\sigma^2}\hat{\beta}.$$

Lastly, we can approximate a solution to the system of linear equations in Algorithm 10 using the Conjugate Gradient Method.

2.6.3 Review of Conjugate Gradient Method

The Conjugate Gradient Method (CGM) is an algorithm for solving a system of linear equations $\boldsymbol{A}x = b$ for $n \times n$ symmetric positive semidefinite matrix $\mathbf{A}$. [26] It is an iterative algorithm, where each iteration has time complexity $O(n^2)$, and it produces an exact solution after n steps, achieving a time complexity $O(n^3)$, equivalent to inverting A. However, one may terminate the algorithm after fewer steps to obtain an approximate solution. In some circumstances, such as those licensed by our application, it may be possible to obtain acceptably close solutions after a number of steps $\tau \ll n$ for a superior time complexity $O(\tau n^2)$.

It is not actually necessary to compute $\boldsymbol{A}$ to perform CGM, only to evaluate its associated linear transformation: and so in our context, if we need to perform a matrix multiplication $\boldsymbol{Q}x$, each party can separately evaluate the matrix multiplication associated with their share of the LD matrix, and these images can be combined. This is formally presented as Algorithm 12.

Algorithm 12 Secure matrix multiplications with $\boldsymbol{Q}$.

```
Input: Public constants $N_i, N$; parameter $\boldsymbol{\Psi^{-1}}$; each party's $\boldsymbol{D}_i$; ciphertext vector $x$.
Output: Ciphertext vector $\boldsymbol{Q}x$.
1: Each party locally computes $\boldsymbol{D}_i x$ using their plaintext $\boldsymbol{D}_i$.
2: The parties aggregate $\sum_{i\in[P]} \frac{N_i}{N}\boldsymbol{D}_i x = \boldsymbol{D}x$.
3: The hub party computes $\boldsymbol{\Psi^{-1}}x$.
4: Return $\frac{N}{\sigma^2}\left(\boldsymbol{D}x + \boldsymbol{\Psi^{-1}}x\right)$.
```

The precision matrices $\boldsymbol{Q}$ that appear in our problem context will always be PSD. The inverse of a PSD matrix is itself PSD, and covariance matrices are always PSD. Consequently, we always satisfy the requirements to deploy CGM.

Convergence guarantees

There are theoretical guarantees on the number of CGM iterations necessary to converge within a particular distance from the optimal solution. These analyses depend upon the *condition number* of the relevant matrix.

Definition 2.6.2 (Condition number)**.** The condition number of a matrix $\mathbf{A}$ with respect to a particular matrix norm $||\cdot||$ is defined as

$$\kappa(\mathbf{A}) := ||\mathbf{A}|| \cdot ||\mathbf{A}||^{-1}.$$

Informally, the condition number of a matrix measures how "well-behaved" the matrix is: how cooperative it will be with algorithms performed upon it. Accordingly, a matrix with a relatively large condition number is called "ill-conditioned," and a matrix with a condition number close to 1 is called "well-conditioned." Concretely, performing CGM upon a well-conditioned matrix will converge to a more accurate solution in fewer iterations. [26]

For interested readers, more information on the condition number, its relationship to CGM, and the (expected) condition number of $\boldsymbol{Q}$ can be found in the Appendix. To summarize the relevant facts, the $\boldsymbol{Q}$ that appear in our problem setting are expected to be ill-conditioned, requiring a number of CGM iterations on the order of the matrix dimension for meaningful convergence. However, there is an effective solution to this issue: preconditioning.

Preconditioning

Preconditioning is a technique to address an ill-conditioned matrix A by instead working with a well-conditioned matrix $M^{-1}A$, where M^{-1} is called a *preconditioner*. [26] If $\kappa(M^{-1}A) \ll \kappa(A)$, then performing CGM on the system $M^{-1}Ax = M^{-1}b$ may converge far quicker than the original system. There are some nuances to this technique ($M^{-1}A$ may no longer be symmetric and PSD) but in summary, it is possible to refactor CGM to implicitly precondition the matrix and achieve a better condition number without needing to explicitly compute $M^{-1}A$. There are many choices of preconditioner matrix M^{-1}, but we use "diagonal preconditioning," also known as "Jacobi preconditioning." The diagonal preconditioner is simply choosing M to be the diagonal matrix whose entries are A's diagonal. This can be easily computed in our context owing to the fact that any SNP will necessarily be in perfect linkage disequilibrium with itself, so the diagonal of $\boldsymbol{D}$ is all ones. So the diagonal preconditioner for us can be easily computed:

$$M = \boldsymbol{\Psi^{-1}} + \boldsymbol{I} \tag{2.9}$$

$$M^{-1} = \operatorname{diag}\left\{\frac{1}{1+\psi_j^{-1}}\right\}_{j\in[M]}. \tag{2.10}$$

The preconditioned conjugate gradient method

These preliminaries established, we present the preconditioned conjugate gradient method as Algorithm 13.

2.6.4 The Conjugate Gradient Method in context

First, I will provide a brief justification of grouping LD blocks together, as at first blush it may seem to worsen runtime. The major determinants of the runtime of CGM are evaluating the matrix-vector products Ax, and the number of iterations before convergence (which is dictated by the condition number $\kappa(A)$.) While both of these operations will generally be faster on smaller matrices, it is nevertheless advantageous to group LD blocks together into single ciphertexts. The runtime of the matrix-vector multiplication algorithm we use is dictated by the *bandwidth* of the matrix A; because aggregating LD blocks forms a block diagonal matrix, aggregating blocks improves the runtime of matrix multiplications. And while grouping LD blocks worsens the condition number, empirically preconditioning causes the condition number to grow very slowly. More detail about these results can be read in the Appendix.

Besides matrix multiplication, most of the operations necessary to implement CGM are straightforward. Inner products can be computed either with HE (using rotations and additions) or MPC, and α/β can be computed with MPC. The primary obstacles to using CGM in the HE/MPC context are instead more structural. First, CGM is known to be sensitive to roundoff error, as occurs in practice when using floating point numbers. [26] This problem becomes more pointed when using HE/MPC due to the limited precision and the perturbations of HE noise. Second, while it's possible to use MPC to check the residual norm δ, strictly speaking terminating early introduces a side channel that can leak information. While there are techniques to decide when to terminate CGM when δ is not available, none of the methods surveyed seemed possible within our context. [27]

While one can instead ignore the residual norm termination condition and only ever terminate after τ iterations, this relies upon an expectation of how the matrix condition

Algorithm 13 Preconditioned Conjugate Gradient Method [26]

Input: Symmetric PSD matrix A, target vector b, preconditioner matrix M^{-1}, maximum number of CGM iterations τ, exact residual update frequency parameter Þ, error tolerance $\varepsilon < 1$.

Output: A vector x_τ such that $Ax_\tau \approx b$.

```
1: Initialize: Set $x \leftarrow \mathbf{0}$, $r \leftarrow b$, $p \leftarrow M^{-1}b$, $\delta_{\text{new}} \leftarrow r^\top p$, $\delta_0 \leftarrow \delta_{\text{new}}$.
2: for $t \in \{1, \ldots, \tau\}$ do
3:     Set $q \leftarrow Ap$.
4:     Set $\alpha \leftarrow \frac{\delta_{\text{new}}}{p^\top q}$.
5:     Set $x \leftarrow x + \alpha p$.
6:     if $t$ is divisible by Þ then
7:         Set $r \leftarrow b - Ax$.
8:     else
9:         Set $r \leftarrow r - \alpha q$.
10:    end if
11:    Set $s \leftarrow M^{-1}r$.
12:    Set $\delta_{\text{old}} \leftarrow \delta_{\text{new}}$.
13:    Set $\delta_{\text{new}} \leftarrow r^\top s$.
14:    if $\delta_{\text{new}} \leq \varepsilon^2 \delta_0$ then
15:        return $x_t$.
16:    end if
17:    Set $\beta \leftarrow \frac{\delta_{\text{new}}}{\delta_{\text{old}}}$.
18:    Set $p \leftarrow s + \beta p$.
19: end for
20: return $x_\tau$.
```

numbers will be distributed over the entire dataset over the entire course of an MCMC run, which is difficult to justify. Empirically it was possible to achieve convergence with $\tau = 8$ iterations. Choosing this parameter is quite sensitive, however, as taking iterations past convergence can cause the algorithm to break: if the residual norm δ becomes too small, there is the risk that it becomes overwhelmed by HE noise, flips to a negative quantity, and incurs undefined behavior in MPC-Divide, which assumes that its denominator is positive. In practice it is worth considering whether the residual norm can be safely leaked so that CGM can be terminated at convergence.

The last concerns with CGM pertain to the tension between managing magnitude and HE noise. If implemented directly in our problem context, the intermediate vectors in CGM might be very small: they are all roughly on the order of the output β or the input vector $b = \eta$. Accordingly, we can scale the input vector by a constant: and actually, by scaling with k_β, we also directly achieve the goal of generating scaled β samples. However, this runs the potential risk that the inner products $p^\top q$ or $r^\top s$ might exceed the upper magnitude limit. Realizing that the inner products only ever appear in ratios, we can address this concern by downscaling the inner products by some known constant, such as $1/d$, where d is the dimension of this LD block. Doing so results in an averaged inner product that should be roughly the same size as the coordinates of the input pointwise product.

2.6.5 The full MVN algorithm

There is a last algorithmic optimization to save a few multiplications, which is slightly changing the system of linear equations to be solved. We would ideally ignore the scaling constant N/σ^2 that appears in the precision matrix, which otherwise will incur multiplications whenever a matrix-vector product is performed.

$$
\begin{aligned}
\boldsymbol{Q}\theta &= z + \boldsymbol{Q}\mu && z \sim \mathrm{MVN}\left(\mathbf{0}, \boldsymbol{Q}\right) \\
\frac{N}{\sigma^2}\left(\boldsymbol{D} + \boldsymbol{\Psi^{-1}}\right)\theta &= \frac{\sqrt{N}}{\sigma} z + \frac{N}{\sigma^2}\hat{\beta} && z \sim \mathrm{MVN}\left(\mathbf{0}, \boldsymbol{D} + \boldsymbol{\Psi^{-1}}\right) \\
(\boldsymbol{D} + \boldsymbol{\Psi^{-1}})\theta &= \frac{\sigma}{\sqrt{N}} z + \hat{\beta} && z \sim \mathrm{MVN}\left(\mathbf{0}, \boldsymbol{D} + \boldsymbol{\Psi^{-1}}\right) \\
(\boldsymbol{D} + \boldsymbol{\Psi^{-1}})\theta' &= \frac{k_\beta \sigma}{\sqrt{N}} z + (k_\beta \hat{\beta}) && z \sim \mathrm{MVN}\left(\mathbf{0}, \boldsymbol{D} + \boldsymbol{\Psi^{-1}}\right)
\end{aligned}
$$

where $\theta' = k_\beta \theta$. Thus we can scale β as desired, perform CGM with the matrix $\boldsymbol{D} + \boldsymbol{\Psi^{-1}}$, without scaling constants; and scale the input vector η for larger CGM intermediates. Putting these components together, we present the revised MVN sampling algorithm as Algorithm 14, which has minor adjustments to manage the magnitude of the intermediate variables.

Algorithm 14 Secure sampling for $k_\beta \beta$.

```
    (For each block $\ell$:)
1: Draw $z \sim \mathrm{MVN}\left(\mathbf{0}, \boldsymbol{D} + \boldsymbol{\Psi}^{\mathbf{-1}}\right)$.
2: Set $\eta = \frac{k_\beta \sigma}{\sqrt{N}} z + k_\beta \hat{\beta}$.
3: Solve $\left(\boldsymbol{D} + \boldsymbol{\Psi}^{\mathbf{-1}}\right) \theta = \eta$ with respect to $x$ using CGM.
4: **return** $\theta$.
```

Chapter 3

Implementation and results

3.1 Implementation

Our secure, federated reimplementation of PRS-CS and the infrastructure for parties to load data and communicate with each other were implemented in the Go programming language. We used the Lattigo library's implementation of CKKS [28] and the Cho lab's "mpc-core" library for additive secret sharing based MPC. [1]

3.2 Results

Sampling algorithm correctness confirmation

As sampling functions are necessarily non-deterministic, we tested the correctness of our implementations by taking a large number of samples and comparing their estimated mean/-variance against the target distribution. The design of our sampling algorithms is specific to the anticipated sizes of the input parameters from the context that these algorithms appear within. Accordingly, to generate realistic values of the parameters we tested upon, we ran the original PRS-CS Python implementation on the provided toy dataset and saved the parameters at an arbitrary iteration. Specifically, the toy dataset uses 1000 SNPs on chromosome 22 and sources its LD matrix from the 1000 Genomes Project European reference panel. [29] We then took 1000 samples of each parameter conditioned upon these fixed parameter values using our secure sampling functions and estimated the mean/variance of the parameter to compare against the target values. For the multivariate parameters, we used the Pearson correlation coefficient to quantify how well the mean/variance matched the target.

In order to benchmark expectations for how much samples can deviate from the target mean/variance and still be considered correct, we also took 1000 samples with a conventional, plaintext sampling function in Python and also plotted these for comparison.

[1]Code available upon request.

Correctness of δ_j sampling

Plots for δ_j sampling are plotted in Figure 3.1. Plaintext samples were taken using NumPy's `random.gamma` function. Samples from our functions are comparable in accuracy to those from the plaintext function. The secure samples' mean has Pearson correlation coefficient 0.999 with the target mean, and their variance has correlation coefficient 0.996 with the target variance. For comparison the plaintext samples' mean has correlation coefficient 0.999 with the target mean, and their variance has correlation coefficient 0.996 with the target variance.

Correctness of σ^2 sampling

As σ^2 is the only univariate distribution, we plot its samples as a histogram in Figure 3.2, and list the estimated mean/variance of the secure and plaintext samples against the target in Table 3.1. The plaintext samples were taken using SciPy's `stats.invgamma`.

	Target	Secure	Plaintext
mean	0.9996	0.9996	0.9995
variance	9.94×10^{-6}	1.02×10^{-5}	1.01×10^{-5}

Table 3.1: Estimated mean/variance of σ^2 samples from both secure and plaintext sampling functions compared with the target.

Correctness of ψ_j sampling

The plots to confirm the correctness of ψ_j sampling are slightly different from the other parameters. Unlike δ_j, σ^2, and β where the target mean/variance can be analytically written and computed, the clamping of the ψ_j complicates efforts to directly compute the target mean/variance. Accordingly, unlike the other distributions, in Figure 3.3 we instead plotted the estimated mean of our secure samples directly against the estimated mean of the plaintext samples; and similarly for the variance. Secure samples were rescaled from $k_\psi \psi_j$ to ψ_j in plaintext, after decrypting. Though a plaintext implementation of the generalized inverse Gaussian was available in SciPy, we opted to instead use the plaintext sampling algorithm implemented in the original PRS-CS codebase owing to differences in parameterization. Our secure sampling algorithm matches the behavior of the plaintext sampling algorithm: the Pearson correlation coefficient between the means is 1.0, and between the variances is 0.998.

Correctness of β sampling

The estimated mean/covariance of samples of β are plotted in Figure 3.4. Secure samples were rescaled from $k_\beta \beta$ to β in plaintext, after decrypting. In order to plot the estimated covariance against the target covariance in a diagonal plot similarly to the mean, the covariance matrix was "flattened" into a 1-D vector using `numpy.ravel`. We observe that our secure, federated algorithm accurately samples from the target distribution, reproducing the faithfulness of a conventional plaintext algorithm, namely NumPy's `random.multivarite_normal`

Parameter	Runtime (seconds)
β	74
σ^2	4
δ_j	3
ψ_j	36

Table 3.2: Approximate runtimes of each sampling function. Times for β, δ_j, ψ_j are for a single ciphertext/group of LD blocks.

function. Moreover, the correlation of our samples' mean/variance against the target is exactly comparable to that of conventional sampling techniques: the secure samples' mean has Pearson correlation coefficient 0.998 with the target mean, and the secure samples' flattened covariance has correlation coefficient 0.895. For comparison, the plaintext samples' mean has correlation coefficient 0.998 with the target mean, and their flattened covariance has correlation coefficient 0.894 with the target covariance.

Runtime estimates

We have conducted preliminary timing experiments on the UK Biobank dataset, specifically for the phenotype low-density lipoprotein and the $\approx 17,000$ SNPs on chromosome 19. [30] Our implementations of the sampling functions lead to about 120 seconds per MCMC iteration, with the particular timing breakdown given in Table 3.2. Note that the 17,000 SNPs fit in three ciphertexts and that sampling can therefore be completely parallelized over them; for larger datasets the time per iteration would likely increase. Nevertheless, these times allow us to roughly extrapolate the anticipated runtime for a full MCMC run on a dataset of size M using a machine C cores, assuming that the LD block sizes permit packing 8000 SNPs on average per ciphertext.

$$1000 \text{ MCMC iterations} \cdot \frac{120 \text{ seconds}}{C \text{ ciphertexts}} \cdot \frac{1 \text{ ciphertext}}{8000 \text{ SNPs}} \cdot M \text{ SNPs} \approx \frac{15M}{C} \text{ seconds}$$

For a full scale study using a dataset of $M = 500,000$ SNPs and a $C = 64$ core machine, this comes out to about a 1.35 day runtime, which is feasible. Note again a key strength of the PRS-CS algorithm: its runtime does not scale in the number of individuals N in the study, collaborations can create large datasets at no additional cost. Also note that communication latency between parties has been omitted, which would increase runtime.

3.3 Future work

The last remaining work is to evaluate the entire PRS-CS algorithm on a full-scale dataset to confirm the accuracy of the sampling functions when combined and confirm our estimates of the algorithm's runtime scaling in the number of SNPs M and number of parties P.

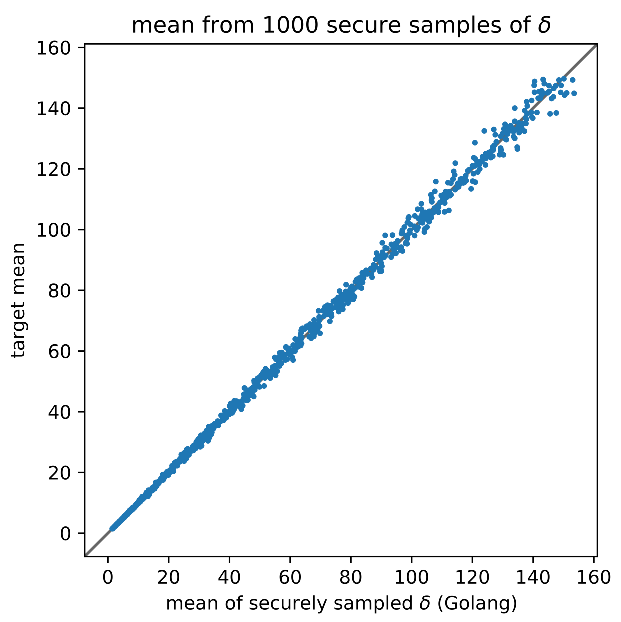
More expansive PRS pipeline

This work focused on the PRS-CS algorithm itself and takes the inputs to the algorithm as exogenous, but these inputs could also be incorporated into the secure computing context. Genetic architecture and linkage disequilibrium effects are understood to differ by ancestry group, and it is therefore important to generate PRS for a particular population from GWAS effect estimates that were also generated from the same population. [1], [14] While our algorithm takes the GWAS effect estimates $\hat{\beta}$ as public, it should be possible to generate them by performing a GWAS on the same dataset used to generate PRS; and indeed the Cho lab has already investigated a secure, federated implementation of GWAS. [7] Algorithmically, it would be trivial to simply mark $\hat{\beta}$ as sensitive.

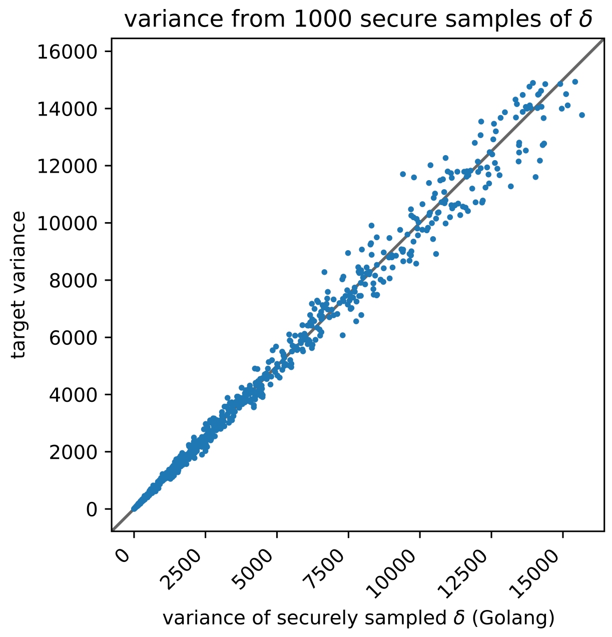
The second key input taken as public are the approximately independent LD block cutoffs. To the extent that LD effects differ between populations, it is reasonable to expect that finding these LD cutoffs specific to the dataset under consideration would improve accuracy. Future work could therefore consider the LD block finding algorithm as part of the entire pipeline and refactor said algorithm under HE/MPC.

3.4 Conclusion

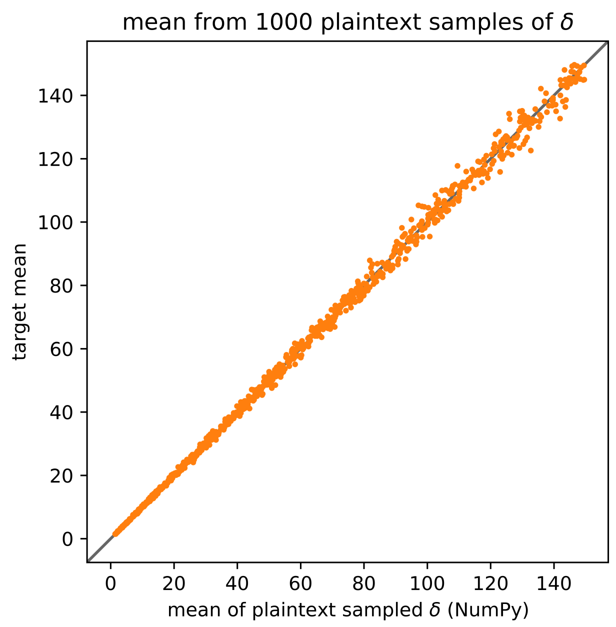
We have exhibited accurate and runtime-practical algorithms for sampling from probability distributions with private parameters to implement PRS-CS within a secure, federated setting. This work will help pave the way for genomics collaborations with larger and more diverse datasets, promising to improve the quality of PRS for a wider range of patients. Furthermore, we expand the explored space of algorithms refactored for privacy preserving multiparty computation and demonstrate the viability of implementing Monte Carlo algorithms within this framework.



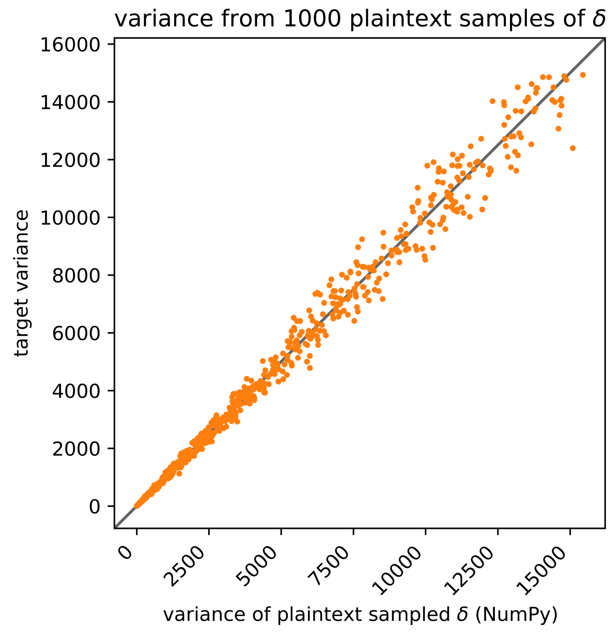
(a) Mean of secure δ_j samples. Pearson correlation coefficient is 0.999.



(b) Variance of secure δ_j samples. Correlation coefficient is 0.996.



(c) Mean of plaintext δ_j samples. Correlation coefficient is 0.999.



(d) Variance of plaintext δ_j samples. Correlation coefficient is 0.996.

Figure 3.1: The estimated mean and variance from 1000 samples of δ_j using our secure sampling function plotted against the target mean/variance (3.1a, 3.1b); and 1000 samples from NumPy's `random.gamma` function plotted against the target mean/variance (3.1c, 3.1d).

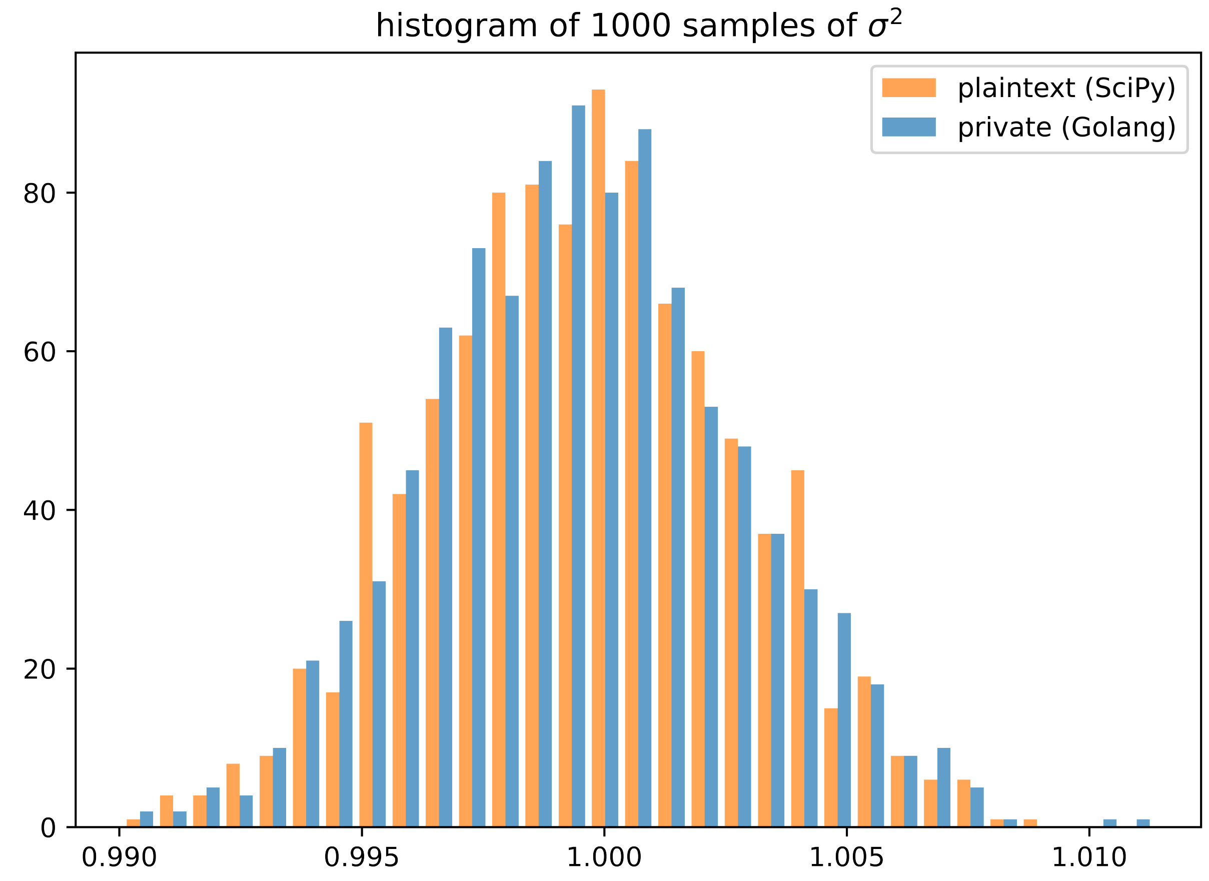
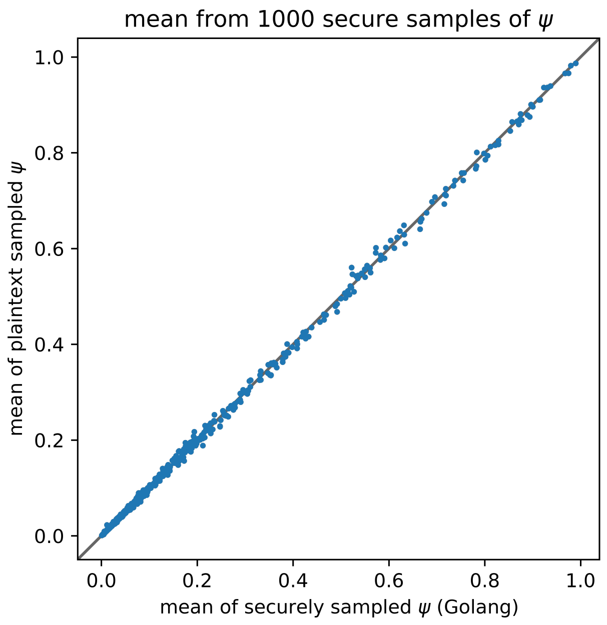
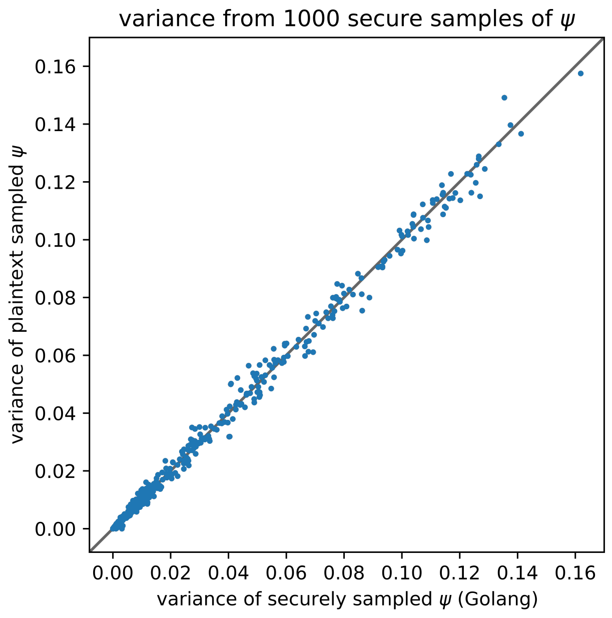


Figure 3.2: A histogram of 1000 samples of σ^2 using our secure sampling function and from SciPy's `stats.invgamma`.

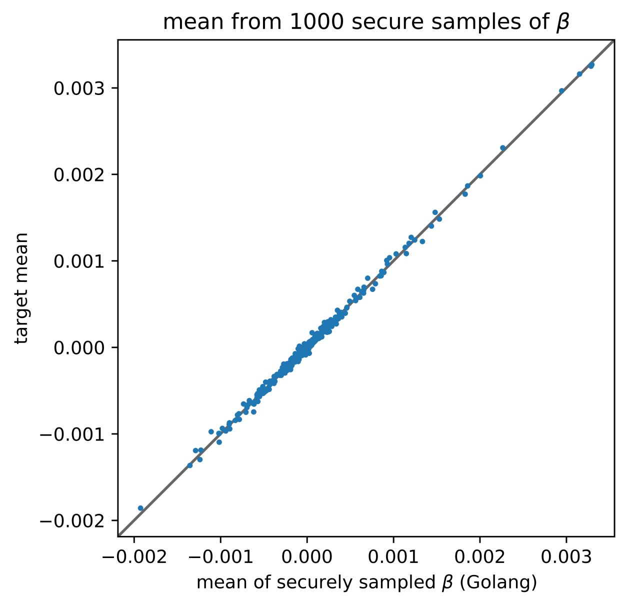


(a)

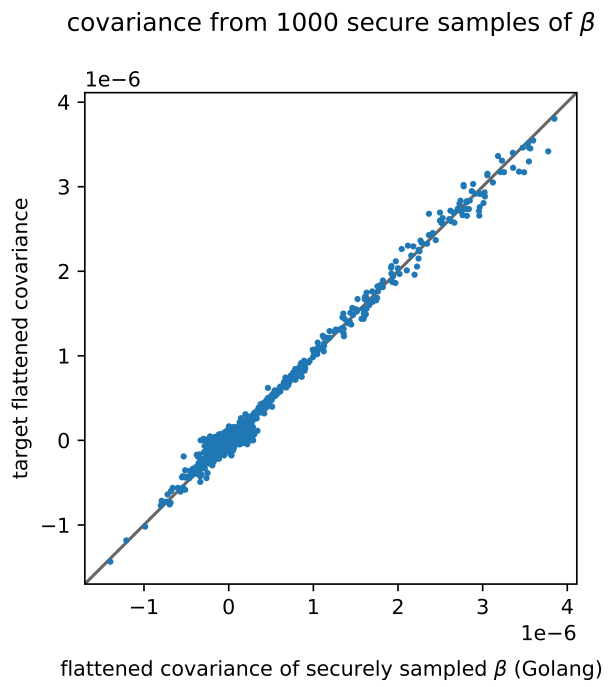


(b)

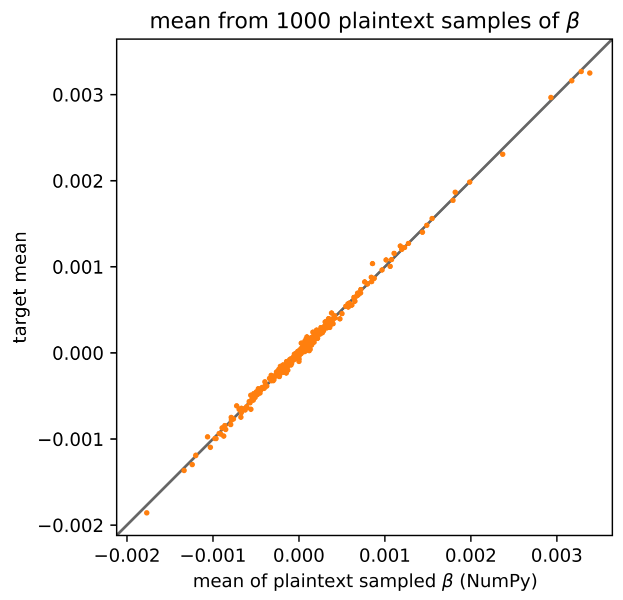
Figure 3.3: The estimated mean (3.3a) and variance (3.3b) from 1000 samples of ψ using our secure sampling function (x axis) and the plaintext sampling function used in PRS-CS (y axis). Pearson correlation coefficient for the mean is 1.0, and for the variance is 0.998.



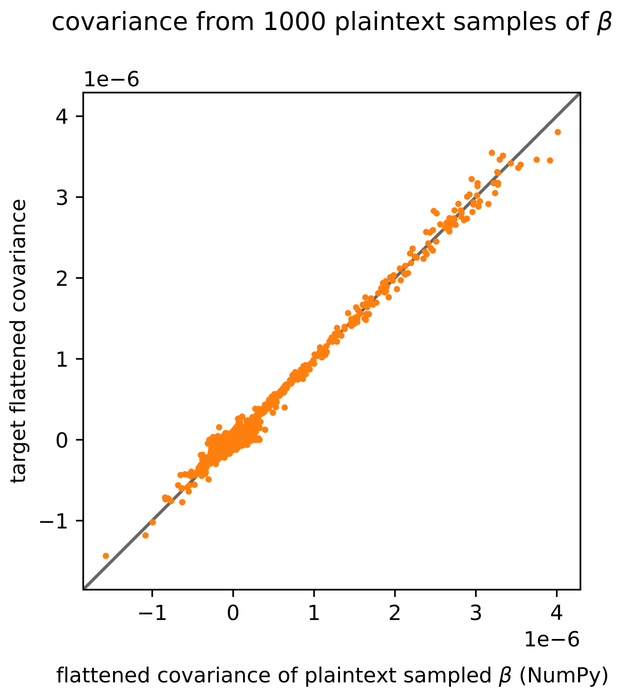
(a) Mean of secure β samples. Pearson correlation coefficient is 0.998.



(b) Flattened covariance matrix of secure β samples. Correlation coefficient is 0.895.



(c) Mean of plaintext β samples. Correlation coefficient is 0.998.



(d) Flattened covariance matrix of plaintext β samples. Correlation coefficient is 0.894.

Figure 3.4: The estimated mean and covariance from 1000 samples of β using our secure sampling function plotted against the target mean/covariance (3.4a, 3.4b); and 1000 samples from NumPy's `random.multivariate_normal` function plotted against the target mean/covariance (3.4c, 3.4d).

References

[1] C. M. Lewis and E. Vassos, "Polygenic risk scores: From research tools to clinical instruments," *Genome Medicine*, vol. 12, 2020. DOI: 10.1186/s13073-020-00742-5.

[2] S. W. Choi, T. S. H. Mak, and P. F. O'Reilly, "A guide to performing polygenic risk score analyses," *Nature Protocols*, 2020. DOI: 10.1038/s41596-020-0353-1. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7612115/.

[3] G. Gürsoy, T. Li, S. Liu, E. Ni, C. M. Brannon, and M. B. Gerstein, "Functional genomics data: Privacy risk assessment and technological mitigation," *Nature Reviews Genetics*, 2021. DOI: 10.1038/s41576-021-00428-7.

[4] Z. Wan, J. W. Hazel, E. W. Clayton, Y. Vorobeychik, M. Kantarcioglu, and B. A. Malin, "Sociotechnical safeguards for genomic data privacy," *Nature Reviews Genetics*, 2022. DOI: 10.1038/s41576-022-00455-y.

[5] K. Zhong, Y. Ma, Y. Mao, and S. Angel, "Addax: A fast, private, and accountable ad exchange infrastructure," 2023. [Online]. Available: https://www.cis.upenn.edu/~sga001/papers/addax-nsdi23.pdf.

[6] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," 2021. [Online]. Available: https://dx.doi.org/10.14722/ndss.2021.24119.

[7] H. Cho, D. Froelicher, J. Chen, M. Edupalli, A. Pyrgelis, J. R. Troncoso-Pastoriza, J.-P. Hubaux, and B. Berger, "Secure and federated genome-wide association studies for biobank-scale datasets," *bioRxiv*, 2022. DOI: 10.1101/2022.11.30.518537. eprint: https://www.biorxiv.org/content/early/2022/12/02/2022.11.30.518537.full.pdf. [Online]. Available: https://www.biorxiv.org/content/early/2022/12/02/2022.11.30.518537.

[8] D. Froelicher, H. Cho, M. Edupalli, J. S. Sousa, J.-P. Bossuat, A. Pyrgelis, J. R. Troncoso-Pastoriza, B. Berger, and J.-P. Hubaux, "Scalable and privacy-preserving federated principal component analysis," in *2023 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 1908–1925. DOI: 10.1109/SP46215.2023.00051. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00051.

[9] H. Cho, D. J. Wu, and B. Berger, "Secure genome-wide association analysis using multiparty computation," *Nature Biotechnology*, vol. 36, 2018. DOI: 10.1038/nbt.4108. [Online]. Available: https://doi.org/10.1038/nbt.4108.

[10] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*. 2023. [Online]. Available: https://toc.cryptobook.us/.

[11] T. Ge, C.-Y. Chen, Y. Ni, Y.-C. A. Feng, and J. W. Smoller, "Polygenic prediction via bayesian regression and continuous shrinkage priors," *Nature Communications*, vol. 10, 2019. DOI: 10.1038/s41467-019-09718-5.

[12] Y. Wang, K. Tsuo, M. Kanai, B. M. Neale, and A. R. Martin, "Challenges and opportunities for developing more generalizable polygenic risk scores," *Annual Review of Biomedical Data Science*, vol. 5, no. 1, pp. 293–320, 2022. DOI: 10.1146/annurev-biodatasci-111721-074830. eprint: https://doi.org/10.1146/annurev-biodatasci-111721-074830. [Online]. Available: https://doi.org/10.1146/annurev-biodatasci-111721-074830.

[13] B. J. Vilhjálmsson, J. Yang, H. K. Finucane, *et al.*, "Modeling linkage disequilibrium increases accuracy of polygenic risk scores," *American Journal of Human Genetics*, 2015. DOI: 10.1016/j.ajhg.2015.09.001.

[14] M. S. Kim, K. P. Patel, A. K. Teng, A. J. Berens, and J. Lachance, "Genetic disease risks can be misestimated across global populations," *Genome Biology*, vol. 19, 2018. DOI: 10.1186/s13059-018-1561-7. [Online]. Available: https://doi.org/10.1186/s13059-018-1561-7.

[15] Y. Lindell, *Tutorials on the Foundations of Cryptography*. Springer, 2017. DOI: 10.1007/978-3-319-57048-8.

[16] J. H. Cheon, A. Kim, M. Kim, and Y. Song, *Homomorphic encryption for arithmetic of approximate numbers*, Cryptology ePrint Archive, Paper 2016/421, https://eprint.iacr.org/2016/421, 2016. [Online]. Available: https://eprint.iacr.org/2016/421.

[17] M. Keller, *Mp-spdz: A versatile framework for multi-party computation*, Cryptology ePrint Archive, Paper 2020/521, https://eprint.iacr.org/2020/521, 2020. DOI: 10.1145/3372297.3417872. [Online]. Available: https://eprint.iacr.org/2020/521.

[18] B. Efron and T. Hastie, *Computer Age Statistical Inference, Algorithms, Evidence, and Data Science*. Cambridge University Press, 2017. [Online]. Available: https://hastie.su.domains/CASI/.

[19] T. Berisa and J. K. Pickrell, "Approximately independent linkage disequilibrium blocks in human populations," 2015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv546.

[20] X. Jiang, M. Kim, K. Lauter, and Y. Song, *Secure outsourced matrix computation and application to neural networks*, Cryptology ePrint Archive, Paper 2018/1041, https://eprint.iacr.org/2018/1041, 2018. DOI: 10.1145/3243734.3243837. [Online]. Available: https://eprint.iacr.org/2018/1041.

[21] L. Devroye, "Random variate generation for the generalized inverse gaussian distribution," *Statistics and Computing*, vol. 24, pp. 236–246, 2014.

[22] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous univariate distributions. Vol. 1, Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics (2nd ed.)* New York: John Wiley & Sons, 1994, pp. 284–285, ISBN: 978-0-471-58495-7.

[23] W. Hoermann and J. Leydold, "Generating generalized inverse gaussian random variates," *Statistics and Computing*, vol. 24, pp. 547–557, 2014.

[24] J. R. Michael, W. R. Schucany, and R. W. Haas, "Generating random variates using transformations with multiple roots," *The American Statistician*, vol. 30, pp. 88–90, 1976.

[25] M. Vono, N. Dobigeon, and P. Chainais, *High-dimensional gaussian sampling: A review and a unifying approach based on a stochastic proximal point algorithm*, 2021. arXiv: 2010.01510 [stat.CO].

[26] J. R. Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, 1994. [Online]. Available: https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.

[27] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. 1994, ISBN: 978-0-89871-328-2. [Online]. Available: https://www.netlib.org/templates/templates.pdf.

[28] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, *Multiparty homomorphic encryption from ring-learning-with-errors*, Cryptology ePrint Archive, Paper 2020/304, https://eprint.iacr.org/2020/304, 2020. DOI: 10.2478/popets-2021-0071. [Online]. Available: https://eprint.iacr.org/2020/304.

[29] T. 1. G. P. Consortium, "A global reference for human genetic variation," *Nature*, 2015. DOI: 10.1038/nature15393.

[30] C. Sudlow, J. Gallacher, N. Allen, V. Beral, P. Burton, J. Danesh, and et al., "Uk biobank: An open access resource for identifying the causes of a wide range of complex diseases of middle and old age," *PLoS Med*, 2015. DOI: 10.1371/journal.pmed.1001779.

Appendix on β sampling

Condition number of block diagonal matrices

First, I will discuss how grouping LD blocks together into a block diagonal matrix affects the expected condition number of the matrices that will be used in CGM.

Lemma 3.4.1 (Norm of a block band matrix)**.** *Let $A_1, \ldots, A_k$ be square matrices along the diagonal of block band matrix D. The norm of D is given as $||D|| = \max\{||A_1||, \ldots, ||A_k||\}$.*

Proof. We use the spectral norm, which is defined as $\rho(A) := \max\{|\lambda_1|, \cdots, |\lambda_n|\}$ for λ_i the eigenvalues of A. By convention, we will number the eigenvalues in the spectrum of a matrix as $|\lambda_1| \geq \cdots \geq |\lambda_k| \geq 0$. We will denote the ith eigenvalue of matrix A_j as λ_i^j. The spectrum of D is the union of the spectra of the A_i, as any eigenvector of an A_i corresponds to an eigenvector of D, with all coordinates not mapping to ones covered by A_i being set to 0. Thus the largest (in magnitude) eigenvalue of D is the largest eigenvalue among A_i, and it belongs to some particular A_i. Then $||D|| = ||A_i||$ for that A_i. $\square$

Lemma 3.4.2 (Condition number of a block band matrix)**.** *Let $\kappa(A_*)$ denote the largest condition number among $\{A_i\}$ in block diagonal matrix D, and likewise let $\kappa(A_{**})$ denote the largest condition number among $\{A_i\} \setminus A_*$, i.e. the second-largest condition number. We can bound the condition number of D as*

$$\sqrt{\kappa(A_*)\kappa(A_{**})} \leq \kappa(D) \leq \kappa(A_*)\kappa(A_{**}).$$

Or, to simplify the result,

$$\kappa(A_*) \leq \kappa(D) \leq \kappa(A_*)^2.$$

Proof. Let us again work with the spectral norm for convenience, and use the same notation as before. Via 3.4.1, we have that

$$\begin{aligned}\kappa(D) &:= ||D|| \cdot ||D^{-1}|| \\ &= \max\{||A_1||, \ldots, ||A_k||\} \cdot \max\{||A_1^{-1}||, \ldots, ||A_k^{-1}||\}.\end{aligned}$$

By the above, clearly $\kappa(D)$ is at least lower bounded by $\max\{\kappa(A_i)\}$. Intuitively, the worst case scenario is therefore one in which there exist *distinct* matrices A_* and A_{**} such that $||A_*||$ and $||A_{**}^{-1}||$ are both very large: we must pick out distinct argmax matrices for each of the factors. And for each of the argmax matrices, in the worst case the matrices are such that $||A_*^{-1}|| \approx 1$ and $||A_{**}|| \approx 1$, such that $||A_*|| \approx \kappa(A_*)$ and $||A_{**}^{-1}|| \approx \kappa(A_{**})$. In this scenario, $\kappa(D)$ would be roughly equal to $\kappa(A_*)\kappa(A_{**})$. $\square$

While the result in Lemma 3.4.2 may look demoralizing, as the condition number can theoretically square, in practice this empirically does not occur. This owes to *preconditioning*, which has the effect of making the matrices such that $||A_i|| \approx ||A_i^{-1}||$. Considering the intuition for the proof of the lemma, it is also intuitive why preconditioning nudges us away from worst-case scenarios, and towards best-case scenarios: if all submatrices are such that $||A_i|| \approx ||A_i^{-1}|| \approx \sqrt{\kappa(A_i)}$, then we will have $\kappa(D) \approx \max\{\kappa(A_i)\}$. In this scenario, we don't actually need to increase the number of CGM iterations at all, or by that much, when packing matrices into block matrices. While we are unable to provide a strong theoretical guarantee against all possible inputs, this scenario is empirically realized under diagonal preconditioning. However, because the condition number generally increases in the block dimension, it is important that groups are all similar in the dimensions of their constituent blocks: a group comprising several large blocks and a group comprising many small blocks will likely have different condition numbers and different CGM convergence times. While a greedy algorithm for assigning blocks to groups sufficed for the datasets we examined, other bin packing techniques might be more even handed for particular datasets.

Matrix multiplication

Matrix multiplications using CKKS ciphertexts can be performed efficiently using a matrix multiplication algorithm of Jiang et al. [8], [20]. In short, to compute AB, that algorithm uses the *diagonals* of A and rotations of the rows of B. Our context poses some additional structure that allows for relatively fast matrix multiplications. Because the LD matrix has banded structure, we can achieve efficient matrix multiplications by skipping multiplication on diagonals outside of the band.

Specifically, we use Algorithm M3 from Froelicher et al. [8], which computes $A \times B$ for an encrypted matrix A and a plaintext matrix B. We can use this algorithm to compute matrix-vector products with the LD matrix $\boldsymbol{D}x$ owing to the symmetry of the LD matrix:

$$\begin{aligned} y &= \boldsymbol{D}x \\ y^\top &= (\boldsymbol{D}x)^\top \\ &= x^\top \boldsymbol{D}^\top \\ &= x^\top \boldsymbol{D}, \end{aligned}$$

thus allowing us to cast the matrix-vector product as a matrix multiplication between a left encrypted "matrix" and a right cleartext matrix. When using this algorithm with a banded cleartext matrix, we can simply omit all computations that would involve diagonals outside the band to drastically improve efficiency. For an $n \times n$ matrix with equal upper and lower bandwidths $k = k_1 = k_2$, we can thus reduce the number of encoded diagonals from $2n - 1$ to $2k + 1$.