Using Homomorphic Encryption for Large Scale Statistical Analysis

David J. Wu and Jacob Haven

Abstract

The development of fully homomorphic encryption schemes in recent years has generated considerable interest in the field of secure computing. In this paper, we consider the problem of performing statistical analysis on encrypted data. Specifically, we focus on two tasks: computing the mean and variance of univariate and multivariate data as well as performing linear regression on a multidimensional, encrypted corpus. Due to the high overhead of homomorphic computation, previous implementations of similar methods have been restricted to small datasets (on the order of a few hundred to a thousand elements) or data with low dimension (generally 1-4). In this paper, we first construct a working implementation of the scale-invariant leveled homomorphic encryption system in [Bra12]. Then, by taking advantage of batched computation as well as a message encoding technique based on the Chinese Remainder Theorem, we show that it becomes not only possible, but computationally feasible, to perform statistical analysis on encrypted datasets with over four million elements and dimension as high as 24. By using these methods along with some additional optimizations, we demonstrate the viability of using leveled homomorphic encryption for large scale statistical analysis.

1 Introduction

In recent years, there has been growing interest in the field of fully homomorphic encryption. Starting from Gentry's breakthrough work in developing the first fully homomorphic encryption (FHE) scheme [Gen09], researchers have proposed many different variants [BV11, BGV12, Bra12] as well as applications [LNV11, GHS12b] of fully homomorphic encryption. At a high level, fully homomorphic encryption allows the evaluation of arbitrary functions on encrypted data. Such a scheme has many applications, particularly in regards to cloud computing and storage platforms. One of the principle concerns people raise regarding cloud-based solutions is the privacy and security of their data. In many domains, such as financial, medical, and military ones, these issues of privacy and confidentiality render cloud-based solutions inappropriate. A natural method to address these concerns would be to store the data encrypted in the cloud. However, if the data is stored encrypted, there must still exist a method of performing useful computations on the encrypted data *without* the need to decrypt the data; in other words, the underlying encryption scheme must be homomorphic.

The concept of homomorphic encryption is not a new one. Soon after the development of the RSA cryptosystem, Rivest, *et al.* [RAD78] introduced the problem of developing an encryption scheme that would support arbitrary manipulations on encrypted data. In the ensuing 30 years, many homomorphic schemes have been proposed, such as the Pailier cryptosystem [Pai99] which supports homomorphic addition of ciphertexts and the ElGamal cryptosystem [EG85], which supports homomorphic multiplication of two ciphertexts. In each of these cases, the scheme is *partially homomorphic* in that they support exactly one homomorphic operation: addition or multiplication. The Boneh-Goh-Nissim cryptosystem [BGN05] is the first scheme to support both homomorphic addition and multiplication. Specifically, this scheme is able to handle an arbitrary number of additions along with a single multiply. Gentry's work [Gen09] presents the first *fully homomorphic* encryption scheme in that it supports the evaluation of an arbitrary number of both additions and multiplications.

In this paper, we consider a leveled homomorphic encryption scheme (without bootstrapping) for statistical analysis over encrypted data. Specifically, we consider computing the mean and covariance of both univariate and multivariate data as well as performing linear regression over encrypted datasets. While there are other methods such as multi-party computation (MPC) protocols that can accomplish the same goal [HFN11], these schemes generally require a high degree of coordination and information exchange between the individual data providers. In the limit where there is a large quantity of data from *multiple* sources, fully homomorphic encryption becomes a more viable solution. To illustrate this, we consider a concrete example. Suppose there are many hospitals providing health records and other potentially sensitive information to a central server. For privacy reasons, this data must be sent encrypted. The central server performs some analysis on the underlying data (i.e., identify certain factors that are strongly correlated with a certain disease or prognosis), and forwards the encrypted result to a researcher. Presumably, the researcher decrypts the ciphertext and uses the results from the analysis to facilitate further investigation. Because our FHE scheme is a public key system, each of the hospitals is able to encrypt and send their data independent of the others. Specifically, we do not require any communication between the hospitals. Thus, when we have a large number of data providers, FHE becomes a more viable method for secure computation.

Using homomorphic encryption schemes to perform statistical analysis is certainly not a new idea. Lauter *et al.* [LNV11] leveraged a somewhat homomorphic encryption scheme to compute the mean and variance of small, univariate datasets, and more recently, Graepel, *et al.* [GLN12] trained a Fisher's linear discriminant classifier over an encrypted dataset. In both of these implementations, the size and dimensionality of the underlying dataset have generally been very small, consisting of several hundred to a thousand elements and working over two to four dimensional data. The principal cause of these limitations has been the significant overhead of fully homomorphic computation. Our contributions in this paper are twofold. First, we present a working implementation of the scale-invariant leveled homomorphic encryption scheme described in [Bra12]. Second, we combine the parallelism from batched computation [SV11, BGV12, GHS12a] with a Chinese Remainder Theorem (CRT) based plaintext representation to enable computation over datasets consisting of several hundred thousand to more than four million data elements. Additionally, we scale up the computation to work over datasets with 24 dimensions in the case of mean and covariance estimation and 5 dimensions in the case of linear regression. In doing so, we demonstrate the possibility of leveraging homomorphic encryption for large scale statistical analysis.

2 Background

2.1 Mathematical Notation

For $q \in \mathbb{Z}$, we identify the ring $\mathbb{Z}/q\mathbb{Z}$ with the integers in $(-q/2, q/2]$. For any $x \in \mathbb{Q}$, we write $[x]_q$ to denote the unique value $y \in (-q/2, q/2]$ such that $x = y \pmod q$. We write $\lfloor x \rceil$ to denote rounding x to the nearest integer and $\lfloor x \rfloor$, $\lceil x \rceil$ to denote rounding down and up to the nearest integer, respectively. In our implementation, we work over polynomial rings modulo a cyclotomic polynomial, $R = \mathbb{Z}[x]/\Phi_m(x)$ where $\Phi_m(x)$ denotes the m^{th} cyclotomic polynomial. We write R_q to denote R/qR. In words, R_q is the set of integer polynomials with degree of at most $\varphi(m) - 1$ and coefficients in $\mathbb{Z}/q\mathbb{Z}$. Finally, unless otherwise specified, the base of the logarithm in our paper is always taken to be 2.

2.2 Canonical Embedding Norm

To facilitate our security and correctness analysis, we need to define a measure on the size of polynomials. Here, we employ the canonical embedding norm as in [LPR10, GHS12a, GHS12b]. For integer m, let ζ_m be a complex primitive m^{th} root of unity. Then, the canonical embedding of $f(x) \in \mathbb{Z}[x]/\Phi_m(x)$ is

given by the $\varphi(m)$-vector of complex numbers $\sigma(f)$ with components $f\left(\zeta^i\right)$ and i ranging over $(\mathbb{Z}/m\mathbb{Z})^*$. The canonical embedding l_p norm of f is defined to be $\|f\|_p^{\text{can}} = \|\sigma(f)\|_p$. In most cases, we will take $p = \infty$. In the following exposition, we make use of several properties of the canonical embedding norm. For all $f, g \in R$, we have that $\|f \cdot g\|_\infty^{\text{can}} \leq \|f\|_\infty^{\text{can}} \|g\|_\infty^{\text{can}}$ and $\|f + g\|_\infty^{\text{can}} \leq \|f\|_\infty^{\text{can}} + \|g\|_\infty^{\text{can}}$. Furthermore, we have $\|f\|_\infty^{\text{can}} \leq \varphi(m) \|f\|_\infty$. Also, there exists a ring constant c_m dependent only on m such that $\|f\|_\infty \leq c_m \cdot \|f\|_\infty^{\text{can}}$. For odd values of m, Damgard, *et al.* [DPSZ11] demonstrated that $c_{2m} = c_m$. Moreover, for primes $p > 11$, $c_p \approx 4/\pi \approx 1.2732$. Finally, we can extend our notions of norm to vectors of polynomials. In particular, given a vector $\mathbf{f} = (f_1, \ldots, f_n) \in R^n$, we define $\|\mathbf{f}\|_p^{\text{can}} = (\sum_{i=1}^p (\|f_i\|_\infty^{\text{can}})^p)^{1/p}$ for $p < \infty$ and $\|\mathbf{f}\|_\infty^{\text{can}} = \max_{1 \leq i \leq p} \|f_i\|_\infty^{\text{can}}$. It follows from the above that given two vectors $\mathbf{f}, \mathbf{g} \in R^n$, $\|\langle \mathbf{f}, \mathbf{g} \rangle\|_\infty^{\text{can}} \leq \|\mathbf{f}\|_\infty^{\text{can}} \|\mathbf{g}\|_1^{\text{can}}$.

2.3 Distributions over R_q

As part of our construction, we will need to sample elements from various distributions over R_q. We describe the relevant distributions below. Note that we can uniquely specify a polynomial $f \in R_q$ by specifying its coefficients. Thus, for simplicity, we describe our distributions over R_q in terms of distributions over $\varphi(m)$-vectors over $\mathbb{Z}/q\mathbb{Z}$. See [GHS12b] for additional information.

- We write $\mathcal{DG}_q(\sigma^2)$ to denote the discrete Gaussian distribution with mean 0 and variance σ^2. We approximate a sample from $\mathcal{DG}_q(\sigma^2)$ by drawing $\varphi(m)$ values from the normal distribution $\mathcal{N}(0, \sigma^2)$ and rounding each value to the nearest integer, reduced modulo q.
- We write $\mathcal{HWT}(h)$ to denote the Hamming weight distribution, defined to be the uniform distribution over the subset of vectors in $\{0, \pm 1\}^{\varphi(m)}$ with exactly h nonzero coefficients.
- We write $\mathcal{ZO}(\rho)$ to denote the distribution over $\{0, \pm 1\}^{\varphi(m)}$ where each element is -1 or 1, each with probability $\rho/2$, and 0 with probability $1 - \rho$.

3 Scale-Invariant Homomorphic Encryption Scheme

We extend the fully homomorphic encryption scheme described in [Bra12] to work over polynomial rings. Along with [BGV12], this is one of the more practical variants of homomorphic encryption. Moreover, the Brakerski system has the advantage in that it does not require modulus switching, and thus, represents a simpler cryptosystem, both in terms of theory and implementation. More precisely, we base security off of the ring learning with errors (RLWE) assumption described in [LPR10]. Here, we describe a simplified version of the RLWE problem that pertains to our implementation. Our presentation is similar to that in [BGV12, LNV11]. For security parameter λ, let $q = q(\lambda), m = m(\lambda)$ be integer functions and let $\chi = \chi(\lambda)$ be a bounded error distribution over $R = \mathbb{Z}[x]/\Phi_m(x)$. Specifically, $|\chi| \leq B$ for some bound B with overwhelming probability. The $\mathsf{RLWE}_{m,q,\chi}$ problem is to distinguish the following two distributions. In the first distribution, sample (a_i, b_i) uniformly from R_q^2. In the second distribution, first sample $s \xleftarrow{\$} R_q$. Then, sample $a_i \xleftarrow{\$} R_q$ and $e_i \xleftarrow{\$} \chi$ and compute $b_i = a_i \cdot s + e_i$. The $\mathsf{RLWE}_{m,q,\chi}$ assumption is that the $\mathsf{RLWE}_{m,q,\chi}$ problem is infeasible.

As in [GHS12b, LNV11], we take our noise distribution χ to be a discrete Gaussian distribution $\mathcal{DG}_q(\sigma^2)$. Instead of drawing the secret key s uniformly from R_q, we instead use very sparse secret keys drawn from $\mathcal{HWT}(h)$. As noted in [GHS12b], using sparse secret keys should not compromise the security of the system. However, this does offer substantial performance improvements as described below.

Finally, besides supporting homomorphic addition and multiplication, we also support homomorphic applications of Frobenius automorphisms. In particular, we can homomorphically map a polynomial $f(x) \in$

R to a polynomial $f^{(i)}(x) \triangleq f(x^i) \pmod{\Phi_m(x)}$. If we let κ_i denote the transformation $\kappa_i : f \mapsto f^{(i)}$, it is well known that the set of transformations $\{\kappa_i \mid i \in (\mathbb{Z}/m\mathbb{Z})^*\}$ forms a group under composition and more specifically, this group is isomorphic to $(\mathbb{Z}/m\mathbb{Z})^*$. [BGV12, GHS12a] demonstrated that if $\mathbf{c}$ is a valid ciphertext encrypting a message m under secret key $\mathbf{s}$, then, $\kappa_i(\mathbf{c})$ is a valid ciphertext encrypting $\kappa_i(m)$ with respect to the secret key $\kappa_i(\mathbf{s})$. Note that $\kappa_i(\mathbf{c})$ and $\kappa_i(\mathbf{s})$ denote the component-wise application of κ_i to $\mathbf{c}$ and $\mathbf{s}$, respectively. With this preparation, we briefly outline our scale-invariant homomorphic encryption (SI-HE) scheme below. Correctness follows as a direct extension of the scheme in [Bra12] to the RLWE setting.

- SI-HE.SecretKeygen(1^λ): Sample $\tilde{\mathbf{s}} \xleftarrow{\$} \mathcal{HWT}_m(h)$. Output $sk = \mathbf{s} = (1, \tilde{\mathbf{s}})$.
- SI-HE.PublicKeygen($\tilde{\mathbf{s}}$): Sample $\mathbf{A} \xleftarrow{\$} R_q$ and $\mathbf{e} \xleftarrow{\$} \mathcal{DG}(\sigma^2)$. Compute $\mathbf{b} := [\mathbf{A} \cdot \tilde{\mathbf{s}} + \mathbf{e}]_q$ and set $\mathbf{P} := [\mathbf{b} \| -\mathbf{A}] \in R_q^2$. Output $pk = \mathbf{P}$.
- SI-HE.Enc$_{pk}(m)$: Given a message $m \in R_p$ and $pk = \mathbf{P}$, sample $\mathbf{r} \xleftarrow{\$} \mathcal{ZO}(0.5)$ and $\mathbf{e} \xleftarrow{\$} \mathcal{DG}^2(\sigma^2)$ and output the ciphertext
$$\mathbf{c} := \left[\mathbf{P}^T\mathbf{r} + p\mathbf{e} + \left\lfloor \frac{q}{p} \right\rfloor \mathbf{m}\right]_q \in R_q^2$$
where $\mathbf{m} \triangleq (m, 0) \in R_p^2$.
- SI-HE.Dec$_{sk}(\mathbf{c})$: Given $\mathbf{c} \in R_q^2$ and $sk = \mathbf{s} = (1, \tilde{\mathbf{s}})$, compute
$$m := \left[\left\lfloor p \cdot \frac{[\langle \mathbf{c}, \mathbf{s} \rangle]_q}{q} \right\rfloor\right]_p.$$
- SI-HE.Add($\mathbf{c}_1, \mathbf{c}_2$): Given two ciphertexts $\mathbf{c}_1, \mathbf{c}_2$ encrypted under the same key $\mathbf{s}$, output $\mathbf{c}_{\text{add}} = \mathbf{c}_1 + \mathbf{c}_2$. Note that if $\mathbf{c}_1$ and $\mathbf{c}_2$ are encrypted under two different secret keys $\mathbf{s}_1, \mathbf{s}_2$, respectively, we can first apply the key switching methods defined below to convert $\mathbf{c}_2$ to a ciphertext $\mathbf{c}_2'$ encrypted under secret key $\mathbf{s}_1$.
- SI-HE.Mult($\mathbf{c}_1, \mathbf{c}_2$): Given two ciphertexts $\mathbf{c}_1, \mathbf{c}_2$ encrypted under secret keys $\mathbf{s}_1, \mathbf{s}_2$, respectively, output
$$\mathbf{c}_{\text{mult}} = \left\lfloor \frac{p}{q} \cdot (\mathbf{c}_1 \otimes \mathbf{c}_2) \right\rceil. \tag{1}$$
The product ciphertext $\mathbf{c}_{\text{prod}}$ is encrypted under the tensored secret key $\mathbf{s}_{\text{mult}} = \mathbf{s}_1 \otimes \mathbf{s}_2$.
- SI-HE.Scalar-Mult($\mathbf{c}, \alpha$). Given ciphertext $\mathbf{c}$ and $\alpha \in R_p$, output the product ciphertext $\mathbf{c}_{\text{scale}} = \alpha \cdot \mathbf{c}$.
- SI-HE.Automorph($\mathbf{c}, \kappa_i$): Given ciphertext $\mathbf{c}$ encrypted under secret key $\mathbf{s}$, output $\mathbf{c}_{\text{auto}} = \kappa_i(\mathbf{c})$. The automorphed ciphertext $\mathbf{c}_{\text{auto}}$ is encrypted under the automorphed secret key $\mathbf{s}_{\text{auto}} = \kappa_i(\mathbf{s})$.

As evidenced above, performing homomorphic operations on ciphertexts will produce ciphertexts encrypted under different secret keys. In the case of homomorphic multiplication, we see that the dimension of the ciphertext and its corresponding secret key squares with each operation. Therefore, it is necessary to provide a method for switching ciphertexts encrypted under one secret key $\mathbf{s}_1$ to a ciphertext encrypted under a different, possibly shorter, secret key $\mathbf{s}_2$. To enable key-switching, we first describe methods of decomposing vectors of ring elements in a way that preserves inner products.

- $\mathsf{BaseDecomp}_{q,b}(\mathbf{x})$: For $\mathbf{x} \in R_q^n$, let $\mathbf{w}_i \in R_b^n$ such that $\mathbf{x} = \sum_{i=0}^{\lceil \log_b q \rceil - 1} b^i \cdot \mathbf{w}_i \pmod q$. Output the vector
$$\left(\mathbf{w}_0, \ldots, \mathbf{w}_{\lceil \log_b q \rceil - 1}\right) \in R_b^{n \cdot \lceil \log_b q \rceil}.$$
- $\mathsf{Powers}_{q,b}(\mathbf{y})$: For $\mathbf{y} \in R_q^n$, output
$$\left[\left(\mathbf{y}, b \cdot \mathbf{y}, \ldots, b^{\lceil \log_b q \rceil - 1} \cdot \mathbf{y}\right)\right]_q \in R_q^{n \cdot \lceil \log_b q \rceil}.$$

It is easy to see that for all $\mathbf{x}, \mathbf{y} \in R_q^n$, $\langle \mathbf{x}, \mathbf{y} \rangle = \left\langle \mathsf{BaseDecomp}_{q,b}(\mathbf{x}), \mathsf{Powers}_{q,b}(\mathbf{y}) \right\rangle$. Now, let $\mathbf{s} \in R_q^{n_s}$ and $\mathbf{t} \in R_q^{n_t}$ be two distinct keys. To switch a ciphertext encrypted under $\mathbf{s}$ to one encrypted under $\mathbf{t}$, we apply the following operations:

- $\mathsf{SwitchKeyGen}_{q,b,\chi}(\mathbf{s}, \mathbf{t})$: Let $\hat{n}_s = (n_s + 1) \cdot \lceil \log_b q \rceil$ denote the dimension of $\mathtt{Powers}_{q,b}(\mathbf{s})$. Sample a uniform matrix $\mathbf{A_{s:t}} \xleftarrow{\$} R_q^{\hat{n}_s \times n_t}$ and a noise vector $\mathbf{e} \xleftarrow{\$} \mathcal{DG}_q^{\hat{n}_s}(\sigma^2)$. Define $\mathbf{b_{s:t}}$ to be
$$\mathbf{b_{s:t}} = \left[\mathbf{A_{s:t}} \cdot \mathbf{t} + \mathbf{e_{s:t}} + \mathsf{Powers}_{q,b}(\mathbf{s})\right]_q \in R_q^{\hat{n}_s}$$
and output
$$\mathbf{P_{s:t}} = [\mathbf{b_{s:t}} \| -\mathbf{A_{s:t}}] \in R_q^{\hat{n}_s \times (n_t + 1)}.$$
- $\mathsf{SwitchKey}_{q,b}(\mathbf{P_{s:t}}, \mathbf{c_s})$: To switch a ciphertext under secret key $\mathbf{s}$ to one under secret key $\mathbf{t}$, compute
$$\mathbf{c_t} := \left[\mathbf{P}_{\mathbf{s:t}}^T \cdot \mathsf{BaseDecomp}_{q,b}(\mathbf{c}_s)\right]_q.$$

4 Statistical Analysis Using FHE

4.1 Statistical Functions

Mean and Covariance. A simple application of our leveled homomorphic encryption scheme is evaluation of the mean and variance of a dataset. The authors of [LNV11] demonstrate the feasibility of using a somewhat homomorphic encryption scheme to compute the mean and variance of small univariate datasets. Here, we extend the work to the multivariate scenario; specifically, given a set of d-dimensional random vectors in $\mathbb{Z}^d$, we estimate both the mean $\mu \in \mathbb{Q}^d$ as well the covariance matrix $\Sigma \in \mathbb{Q}^{d \times d}$. By batching the data (described below), we demonstrate that it is feasible to evaluate the mean and covariance of large datasets with upwards of 10^6 elements as well as of datasets with dimension as high as 24 (in which case the covariance matrix consists of 300 unique entries). In addition to enabling multivariate statistical estimation, the covariance matrix also captures the relation and correlation between different variables in the dataset. It also forms the basis of more sophisticated statistical techniques such as principal component analysis (PCA).

We briefly describe our approach. Consider a dataset $\mathcal{D} = \left\{x^{(i)} \mid i = 1, \ldots, n\right\}$ where each $x^{(i)} \in \mathbb{Z}^d$. Specifically each data element is a random vector $x^{(i)} = \left[x_1^{(i)}, \ldots, x_d^{(i)}\right]^T$. To compute the mean $\mu \in \mathbb{Q}^d$ we simply compute $\mu = \frac{1}{n} \sum_{i=1}^n x^{(i)}$. Note that since our homomorphic encryption scheme does not support division efficiently, we return the numerator and denominator separately. Next, we consider the covariance. By definition, the covariance of two random variables X, Y is given by $\mathrm{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$. Then, letting Σ_X denote the covariance matrix of X, we have
$$(\Sigma_X)_{ij} = \mathrm{Cov}(X_i, X_j) = \mathbb{E}\left[X_i X_j\right] - \mathbb{E}\left[X_i\right]\mathbb{E}\left[X_j\right] = \frac{1}{n}\sum_{k=1}^n x_i^{(k)} x_j^{(k)} - \frac{1}{n^2}\left(\sum_{k=1}^n x_i^{(k)}\right)\left(\sum_{k=1}^n x_j^{(k)}\right).$$

We rewrite this as a matrix product. First, define the matrix $X \in \mathbb{Z}^{n \times d}$ where the i^{th} row of X consists of the i^{th} example $\left[x^{(i)}\right]^T$. Then, it is easy to see that the covariance matrix is just given by

$$\Sigma_X = \frac{1}{n} X^T X - \frac{1}{n^2} (n\mu)(n\mu)^T = \frac{1}{n^2} \left(n X^T X - (n\mu)(n\mu)^T \right). \tag{2}$$

Note that we can reuse the value of $n\mu$ obtained by computing the mean. It is worth noting that both mean and covariance computations may be performed using circuits with low depth

Linear Regression. In addition to multivariate analysis, we consider another common statistical estimation problem: linear regression. Let $\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) \mid i = 1, \ldots, n \right\}$ be a dataset with input variables $x^{(i)} \in \mathbb{Z}^d$ and output variables $y^{(i)} \in \mathbb{Z}$. In linear regression, our goal is to approximate y as a linear function $h_\theta(x) = \theta^T x$ of x where $\theta \in \mathbb{Q}^d$ is a set of parameters. We choose the parameters θ so as to minimize the least-squares regression error; specifically, we compute

$$\theta^\star = \left(X^T X \right)^{-1} X^T y, \tag{3}$$

where X is the design matrix and y is the vector of response variables:

$$X = \begin{bmatrix} \text{—} \left(x^{(1)} \right)^T \text{—} \\ \text{—} \left(x^{(2)} \right)^T \text{—} \\ \vdots \\ \text{—} \left(x^{(n)} \right)^T \text{—} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Thus, we can easily perform linear regression using a series of matrix-matrix products and matrix-vector products. The only complication is the computation of the matrix inverse. For small dimension d, we leverage Cramer's rule. More precisely, we have that

$$\left(X^T X \right)^{-1} = \frac{1}{\det\left(X^T X \right)} \text{Adj}\left(X^T X \right)$$

where $\text{Adj}(M)$ denotes the adjugate of matrix M. As before, we separately compute the numerator and denominator of $\theta^\star$.

4.2 Data Representation and Batching

The efficiency of homomorphic evaluation of arithmetic circuits depends significantly on our choice of message encoding. Lauter, *et al.* [LNV11] describe two methods of encoding integers in R_p. The first and most natural method is to encode the integers as constant polynomials in R_p. However, given that elements of R_p are polynomials of degree $\varphi(m)$, this is not an efficient encoding scheme. An alternative encoding scheme is to use a bitwise encoding of each integer. For example, if we let $b_0, b_1, \ldots, b_k$ be the bitwise representation of $m \in \mathbb{Z}$, then we can represent m with the polynomial $\sum_{i=0}^k b_i x^i$. Addition and multiplication of two integers encoded in this manner translates to just polynomial addition and multiplication in R_p. As long as the coefficients of the resulting polynomials do not exceed p and their degrees do not exceed $\varphi(m)$, correctness is ensured and we can we can recover the message by evaluating the polynomial at $x = 2$. These constraints, however, render this scheme impractical for even circuits of moderate depth. To support computations involving many large integers, we would require prohibitively large values of m and p under this representation.

To construct a scheme that supports computations over $\mathbb{Z}/p\mathbb{Z}$ for some large value p, we leverage the Chinese Remainder Theorem (CRT). In our case, we take $p > 2^{128}$ so our scheme supports at least 128-bit computation. More precisely, we take p to be a product of many small prime factors: $p = \prod_{i=1}^k p_i$

where $p_1, \ldots, p_k$ are distinct primes. To evaluate an arithmetic circuit over the elements of $\mathbb{Z}/p\mathbb{Z}$, we evaluate the circuit over the values modulo each prime $p_1, \ldots, p_k$. Using the results from our evaluation over $\mathbb{Z}/p_1\mathbb{Z}, \ldots, \mathbb{Z}/p_k\mathbb{Z}$, we recover the result over $\mathbb{Z}/p\mathbb{Z}$ by application of the CRT. By varying k, we can restrict our computations to smaller moduli at the expense of having to perform more computations. In our experiments, we generally take $k \approx 10$, in which case we have $p_i \approx 2^{12}$ for $i = 1, \ldots, k$.

To enable computations over large amounts of data, we pack multiple plaintext messages into one ciphertext block. Specifically, as observed by Smart and Vercauteren [SV11], the plaintext space R_p can be broken up into a vector of "plaintext slots" by application of the polynomial CRT. While the ring modulus $\Phi_m(x)$ is irreducible over $\mathbb{Z}$, it might be reducible over $\mathbb{Z}/p\mathbb{Z}$ for some p. Then, $\Phi_m(x) = \prod_{i=1}^{\ell} F_i(x) \pmod p$, where each of the $F_i(x)$ is an irreducible polynomial modulo p. Each factor in this decomposition has equal degree d where d is the smallest value such that $p^d = 1 \pmod m$. Since $\Phi_m(x)$ has degree $\varphi(m)$, we have that $\ell \cdot d = \varphi(m)$. Finally, note that when $p \equiv 1 \pmod m$, we automatically have $d = 1$ and so $\Phi_m(x)$ splits into $\varphi(m)$ linear factors. When $\Phi_m(x)$ reduces modulo p, the plaintext space decomposes according to $R_p = \bigotimes_{i=1}^{\ell} R_{\mathfrak{p}_i}$ where $\mathfrak{p}_i$ is the ideal in R generated by p and $F_i(x)$. Thus, by the polynomial CRT, the plaintext space decomposes into ℓ non-interacting slots. Then, given ℓ messages $m_1 \in R_{\mathfrak{p}_1}, \ldots, m_\ell \in R_{\mathfrak{p}_\ell}$, we can pack them into a single message $m \in R_p$. If we have two packed vectors m, m' corresponding to messages $m_1, \ldots, m_\ell$ and $m'_1, \ldots, m'_\ell$, respectively, computing the product $m \cdot m'$ yields a packed plaintext with messages $m_1 \cdot m'_1 \in R_{\mathfrak{p}_1}, \ldots, m_\ell \cdot m'_\ell \in R_{\mathfrak{p}_\ell}$. Similarly, $m + m'$ translates to an element-wise sum. Hence, by batching ℓ messages into a single plaintext block, it becomes possible to perform ℓ additions or multiplications at the cost of just a single operation.

Both linear regression and covariance computation can be performed using a series of matrix products, which in turn reduces to computing a series of inner products. Consider two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^n$. To compute $\langle \mathbf{u}, \mathbf{v} \rangle$, we compute $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{n} u_i v_i$. Notice that if we pack the elements of $\mathbf{u}$ into one plaintext block $\tilde{\mathbf{u}}$ and the elements of $\mathbf{v}$ into another plaintext block $\tilde{\mathbf{v}}$, we can evaluate all n products $u_i v_i$ using a single homomorphic multiply. When the batch size is large, this significantly reduces the number of multiplications we need to evaluate, which by corollary, enables support for computation over significantly larger datasets. The problem, however, is evaluating the sum of the elements within the packed ciphertext block. Here, we leverage the observation made in [BGV12, GHS12a], who showed that it was possible to rotate or permute the underlying plaintext slots in a batched vector by applying automorphisms associated with the ring R. First, consider an automorphism κ which simply rotates the plaintext slots in a message m. Specifically, if m is a block with ℓ slots denoted $(m_1, m_2, \ldots, m_\ell)$, then $\kappa(m)$ would correspond to the block with slots $(m_2, m_3, \ldots, m_\ell, m_1)$. Let $k = \lfloor \log \ell \rfloor$. We can compute $\sum_{i=1}^{2^k} m_i$ using k rotations as follows:

1. Set $m' \leftarrow m$.
2. For $i = 0, \ldots, k = \lfloor \log \ell \rfloor$,
   (a) Set $m' \leftarrow m' + \kappa^{2^i}(m')$.
3. Return m'. The element in the first slot m'_1 is the desired sum $\sum_{i=1}^{2^k} m_i$.

It is not difficult to see that this method computes the desired value of m'. When ℓ is not an even power of two, we note that the messages in the remaining slots leak information about the values in m. To address this problem, we add a uniformly random value in $\mathbb{Z}/p\mathbb{Z}$ to all but the first slot. In doing so, we ensure that the values in the remaining slots are uniformly random and thus, leak no additional information about the original values in m. Using, this method, we are able to evaluate dot products over batched vectors.

From the above discussion, we see that as long as there exists an automorphism κ that rotates the plaintext slots, we can evaluate the sum of the first 2^k elements in a given block. In fact, it suffices

that there exists κ that permutes the slots according to some permutation π such that π has exactly one cycle. As noted in [GHS12a], it is then possible to reorder the slots (i.e., the CRT factors $F_i(x)$) such that κ produces a rotation of the slots. In particular, we fix a value for the first factor $F_1(x)$ and then reorder the remaining factors $F_i(x)$ according to $\pi^{i-1}(1)$. We consider conditions under which such an automorphism exists. Recall that the set of transformations $\{\kappa_i \mid i \in (\mathbb{Z}/m\mathbb{Z})^*\}$ under composition is the Galois group $\mathcal{G}\mathsf{al}\,(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ where $\mathbb{Q}(\zeta_m)$ denotes the m^{th} cyclotomic number field. Moreover, $\mathcal{G}\mathsf{al}\,(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ is isomorphic to $(\mathbb{Z}/m\mathbb{Z})^*$. We know that $(\mathbb{Z}/m\mathbb{Z}^*)$ is cyclic if $m = p^k$ or $m = 2p^k$ for odd primes p and $k > 0$. Let g be a generator for $(\mathbb{Z}/m\mathbb{Z})^*$. Because $(\mathbb{Z}/m\mathbb{Z})^*$ is isomorphic to $\mathcal{G}\mathsf{al}\,(\mathbb{Q}(\zeta_m)/\mathbb{Q})$, the automorphism κ_g will produce a cyclic permutation of the plaintext slots. In our implementation, we choose $m = 2p$ for some prime p.

We conclude by illustrating how this representation significantly reduces the computational cost of evaluating matrix products. Consider the design matrix $X \in \mathbb{Z}^{n\times d}$ from linear regression where n is the number of examples and d is the dimension of each example. Let ℓ denote the number of plaintext messages we can pack into a single ciphertext. In our batched matrix representation, we break each column of X into $\lceil \frac{n}{\ell} \rceil$ individual blocks, with each block containing at most ℓ elements. Each block of data maps to a single ciphertext; specifically, the block matrix has dimension $\lceil \frac{n}{\ell} \rceil \times d$. Thus, when we compute the product of two matrices, we perform block matrix multiplication rather than standard matrix multiplication. It is worth noting, however, that the cost of multiplying two ciphertext blocks is independent of the number of plaintext elements within those blocks. Thus, evaluating $X^T X$ just requires $\lceil \frac{n}{\ell} \rceil d^2$ multiplications as opposed to nd^2, which is a considerable improvement for large ℓ.

4.3 Implementation Details

As described earlier, our implementation is based on the RLWE variant of the scale-invariant leveled homomorphic scheme described in [Bra12]. We consider two possible ways of representing polynomials in R_q. The most intuitive way of representing a polynomial $f(x) \in R_q$ is through a vector of $\varphi(m)$ coefficients. Specifically, if $f(x) = \sum_{i=0}^{\varphi(m)-1} a_i x^i$ with $a_i \in \mathbb{Z}/q\mathbb{Z}$, then the *coefficient representation* of f is simply the vector $(a_0, \ldots, a_{\varphi(m)-1}) \in (\mathbb{Z}/q\mathbb{Z})^{\varphi(m)}$. An alternative representation is to take the values of the polynomial at the primitive m^{th} roots of unity modulo q. Supposing that q contains a primitive m^{th} root of unity $\zeta \in \mathbb{Z}/q\mathbb{Z}$, the primitive m^{th} roots of unity modulo q are simply the elements of the set $\{\zeta^i \mid i \in (\mathbb{Z}/q\mathbb{Z})^*\}$. Thus, the *evaluation representation* of f is a vector of $\varphi(m)$ components, each corresponding to $f(\zeta^i)$ for some $i \in (\mathbb{Z}/q\mathbb{Z})^*$. Finally, we note that if q is composite, we can leverage the CRT and represent each of the evaluations relative to the factors of q. Following the convention and definitions in [GHS12b], we refer to this as the *double CRT* representation.

We briefly consider the tradeoffs between the two representations. Multiplying two degree d polynomials requires time $O(d^2)$ when the polynomials are in coefficient representation, but just time $O(d)$ when they are in evaluation representation. We can convert between the coefficient and evaluation representations by leveraging the Fast Fourier Transform (FFT) and inverse FFT algorithms which require time $O(d \log d)$. Thus, in virtually all cases, it is advantageous to convert from coefficient representation to evaluation representation to carry out the polynomial multiplications. In fact, Gentry *et al.* [GHS12b] keep their polynomials in double CRT representation throughout the homomorphic evaluation process, only converting back to coefficient representation if absolutely critical. In the case of Brakerski's scale-invariant system [Bra12], this does not appear to be a viable option. The complication arises from the scale invariant nature of the scheme. Then, when we multiply two ciphertexts as prescribed by Equation (1), it is necessary to rescale the tensored ciphertext; this procedure requires that we round each coefficient in the polynomial to the nearest integer. At this time, we are not aware of a way to round polynomials in

evaluation representation and as such, we perform the rounding steps in coefficient representation.[1] Thus, in contrast to the BGV-based implementation in [GHS12b], our implementation will require explicit conversions between coefficient representation and evaluation representations. Since performing many FFT and inverse FFTs are expensive operations and can quickly dominate the cost of homomorphic evaluation, we try to minimize the number of times we need to convert between the two representations.

Since we perform rounding in coefficient representation, we use coefficient representation as the base representation of our polynomials. To simplify the arithmetic involved in division and rounding, we take our modulus q to be a power of two. In this case, we can divide and round using simple bitwise operations. When forming the tensor product between two ciphertexts, we first convert the polynomials to double CRT representation over a sufficiently large modulus. Specifically, we note that the tensoring operation $\mathbf{c}_1 \otimes \mathbf{c}_2$ in Equation (1) is done over R and not R_q. As such, we can construct a lower bound on the size of the necessary double CRT modulus. Given two polynomials $f, g \in R_q$,

$$\|fg\|_\infty \leq c_m \|fg\|_\infty^{\text{can}} \leq c_m \|f\|_\infty^{\text{can}} \|g\|_\infty^{\text{can}} \leq c_m \varphi(m) \|f\|_\infty \varphi(m) \|g\|_\infty \leq c_m \varphi^2(m) q^2.$$

To compute $p(\mathbf{c}_1 \otimes \mathbf{c}_2)$ without overflow, we thus take our double CRT modulus to be at least $c_m p q^2 \varphi^2(m)$ when evaluating the tensor. Upon forming the tensor, we convert back to coefficient representation, divide, and round accordingly.

We now describe some optimizations specific to evaluating matrix products that reduces the number of times we need to convert between coefficient and evaluation representations and the number of times we have to key switch. Let $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_\xi)$ and $\mathbf{c}' = (\mathbf{c}'_1, \ldots, \mathbf{c}'_\xi)$ be two vectors of ciphertexts, each encrypted under $\mathbf{s}$. For instance, these may correspond to the encryption of two rows of a matrix. Consider computing the dot product of $\mathbf{c}$ and $\mathbf{c}'$. Naively applying Equation (1) and key switching after each multiplication, this translates to computing

$$\langle \mathbf{c}, \mathbf{c}' \rangle = \sum_{i=1}^{\xi} \mathsf{SwitchKey}_{q,b} \left(\mathbf{P}_{\mathbf{s} \otimes \mathbf{s}:\mathbf{s}}, \left\lfloor \frac{p}{q} \cdot (\mathbf{c}_i \otimes \mathbf{c}'_i) \right\rceil \right).$$

First, we note that there is no need to key switch after computing $\mathbf{c}_i \otimes \mathbf{c}'_i$. Since the cost of adding two tensored ciphertexts is much less than the cost of key switching, it is more efficient to add the tensored ciphertexts and apply the key switch to the final result. Furthermore, we note that though we need to compute ξ pairwise products, there is no reason to convert back to coefficient representation and round after each product. Instead, we compute the sum of the tensored ciphertexts and rescale and round only once. Thus, to compute the dot product more efficiently, we instead evaluate

$$\langle \mathbf{c}, \mathbf{c}' \rangle = \mathsf{SwitchKey}_{q,b} \left(\mathbf{P}_{\mathbf{s} \otimes \mathbf{s}:\mathbf{s}}, \left\lfloor \frac{p}{q} \sum_{i=1}^{\xi} (\mathbf{c}_i \otimes \mathbf{c}'_i) \right\rceil \right).$$

Finally, as noted above, we ensure that the double CRT modulus does not wrap around by setting the modulus to be at least $c_m \xi p \varphi^2(m) q^2$.

Last, we make a brief note regarding our choice of values for the plaintext modulus p. Recall that we want $p = 1 \pmod m$ so $\Phi_m(x)$ splits into linear factors modulo p. Furthermore, in order to perform rotations of the plaintext slots, we require that $m = q^k$ or $m = 2q^k$ for some prime q and $k > 0$. A natural choice then is to let p be a safe prime (i.e., $p = 2q + 1$ for some prime q) and take $m = p - 1 = 2q$. Clearly, this choice of m and p trivially satisfies our conditions. As discussed in Section 2.2, this particular choice of m has the added benefit that the ring constant c_m is bounded by a small constant, namely $4/\pi \approx 1.2731$.

[1] It might be possible to reduce the number of FFTs we perform by using the Scale function described in [GHS12b]. However, in this case, we cannot choose our ciphertext modulus q to be a power of two, which complicates the scaling and rounding procedures. We did not test this alternative implementation.

5 Security Analysis and Parameter Settings

5.1 Canonical Embedding Norm for Distributions

Following the discussion in [GHS12b], we bound the canonical embedding of polynomials sampled from our various distributions. Let $f(x) \in R$ be a polynomial sampled from some distribution over R. Suppose further that each coefficient in f is drawn iid from some distribution with mean 0 and variance σ_f^2 (this assumption holds for all of the distributions we are considering). Then, if we denote $f(x) = \sum_{i=0}^{\varphi(m)-1} a_i x^i$, we have that $f(\zeta) = \sum_{i=0}^{\varphi(m)-1} a_i \zeta^i$, which is a sum of random variables. Since the coefficients a_i are iid with mean 0 and variance σ^2, the mean $\mathbb{E}[f(\zeta)]$ and variance $\mathrm{Var}[f(\zeta)]$ are given by

$$\mathbb{E}[f(\zeta)] = \sum_{i=0}^{\varphi(m)-1} \zeta^i \mathbb{E}[a_i] = 0$$

$$\mathrm{Var}[f(\zeta)] = \sum_{i=0}^{\varphi(m)-1} \left(\zeta^i\right)^2 \mathrm{Var}[a_i] = \sigma_f^2 \sum_{i=0}^{\varphi(m)-1} \left(\zeta^i\right)^2 = \sigma_f^2 \varphi(m).$$

Finally, we note that $f(\zeta)$ is the sum of many iid random variables and so, by the Central Limit Theorem, $f(\zeta)$ is approximately normally distributed with mean 0 and variance $\sigma_f^2 \varphi(m)$. As in [GHS12b], we use 6σ as a high probability bound on the magnitude of $f(\zeta)$. Now, we consider our particular distributions. If $f \sim \mathcal{DG}(\sigma^2)$, then $\sigma_f^2 = \sigma^2$, so we have $\|f\|_\infty^{\mathrm{can}} \leq 6\sigma\sqrt{\varphi(m)}$. If $f \sim \mathcal{ZO}(\rho)$, $\sigma_f^2 = \rho$, which yields $\|f\|_\infty^{\mathrm{can}} \leq 6\sqrt{\rho\varphi(m)}$. Finally, if $f \sim \mathcal{HWT}(h)$, f has only h nonzero components so $\mathrm{Var}[f(\zeta)] = h$. Moreover, $\|f\|_\infty^{\mathrm{can}} \leq 6\sqrt{h}$. There will also be cases where we have to bound the canonical embedding norm of the product of two randomly sampled polynomials f, g. In this case, we need to bound the magnitude of the product of two random variables. If the coefficients of f, g have variances σ_f^2 and σ_g^2, respectively, then we use $16\sigma_f\sigma_g$ as our bound for $\|fg\|_\infty^{\mathrm{can}}$ as in [GHS12b].

5.2 Error Bounds for Homomorphic Operations

Given the above preparation, we can analyze the amount of noise introduced by each of our operations. Consider a ciphertext $\mathbf{c}$ encrypting a message m under secret key $\mathbf{s} = (1, \tilde{\mathbf{s}})$. Note also that $\mathbf{m} = (m, 0)$ and $\mathbf{e} = (e_0, e_1)$. By construction,

$$\begin{aligned}
\langle \mathbf{c}, \mathbf{s} \rangle &= \left\lfloor \frac{q}{p} \right\rfloor m + \left(\mathbf{r}^T \mathbf{P} + p\mathbf{e}\right) \cdot (1, \tilde{\mathbf{s}}) \pmod q \\
&= \left\lfloor \frac{q}{p} \right\rfloor m + \mathbf{r} \cdot \mathbf{b} - \mathbf{r} \cdot \mathbf{A}\tilde{\mathbf{s}} + pe_0 + pe_1\tilde{\mathbf{s}} \pmod q \\
&= \left\lfloor \frac{q}{p} \right\rfloor m + \underbrace{\mathbf{r} \cdot \mathbf{e} + p\left(e_0 + e_1\tilde{\mathbf{s}}\right)}_{E} \pmod q. \qquad (4)
\end{aligned}$$

where E denotes the noise in the ciphertext. By construction of the decryption function, decryption will succeed if $\|E\|_\infty < \lfloor q/2 \rfloor / p$. Since $\|E\|_\infty \leq c_m \|E\|_\infty^{\mathrm{can}}$, it is sufficient to require that

$$\|E\|_\infty^{\mathrm{can}} \leq \frac{\lfloor q/2 \rfloor}{c_m \cdot p}. \qquad (5)$$

Thus, we consider $\mathbf{c}$ to be a valid encryption of a message m if both (4) and (5) hold. Now, consider the noise from a fresh encryption. From the bounds we computed above for randomly sampled polynomials,

we have that the norm of the initial noise $E_{\text{clean}} = \mathbf{r} \cdot \mathbf{e} + p(e_0 + e_1\tilde{\mathbf{s}})$ is bounded by

$$\begin{aligned}\|E_{\text{clean}}\|_\infty^{\text{can}} &\leq \|\mathbf{r} \cdot \mathbf{e}\|_\infty^{\text{can}} + p\,\|e_0\|_\infty^{\text{can}} + p\,\|e_1\tilde{\mathbf{s}}\|_\infty^{\text{can}} \\ &\leq 8\sqrt{2}\sigma\varphi(m) + 6\sigma p\sqrt{\varphi(m)} + 16\sigma p\sqrt{h \cdot \varphi(m)}. \end{aligned} \tag{6}$$

Now, we proceed to analyze the amount of noise generated by each of our operations.

Key Switching. For ciphertext $\mathbf{c_s}$ and $\mathbf{c_t}$ encrypted under secret keys $\mathbf{s} = (1, \tilde{\mathbf{s}})$ and $\mathbf{t} = (1, \tilde{\mathbf{t}})$ respectively, we have that

$$\langle \mathbf{c_t}, \mathbf{t} \rangle = \langle \mathbf{c_s}, \mathbf{s} \rangle + \left\langle \mathsf{BaseDecomp}_{q,b}(\mathbf{c_s}), \mathbf{e_{s:t}} \right\rangle \pmod q.$$

The additive noise introduced by key-switching is given by

$$\begin{aligned}\left\|\left\langle \mathsf{BaseDecomp}_{q,b}(\mathbf{c_s}), \mathbf{e_{s:t}} \right\rangle\right\|_\infty^{\text{can}} &\leq \left\|\mathsf{BaseDecomp}_{q,b}(\mathbf{c}_s)\right\|_\infty^{\text{can}} \|\mathbf{e_{s:t}}\|_1^{\text{can}} \\ &\leq [b\varphi(m)] \cdot \left[(n_s+1)\left\lceil \log_b q \right\rceil \cdot 6\sigma\sqrt{\varphi(m)}\right] \\ &= 6\sigma b(n_s+1)\left\lceil \log_b q \right\rceil \varphi^{3/2}(m)\end{aligned}$$

where n_s is the dimension of $\mathbf{s}$ ($n_s = 2$ if we are switching from automorphed keys and $n_s = 4$ if we are switching from tensored keys). With this preparation, we are ready to compute the noise introduced by each of the homomorphic operations. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be two ciphertexts encrypting messages m_1 and m_2 under the *same* secret key $\mathbf{s}$. Denote the noise in the encryptions by E_1 and E_2, respectively.

Addition. For addition, we have,

$$\langle \mathbf{c}_{\text{add}}, \mathbf{s} \rangle = \langle \mathbf{c}_1, \mathbf{s} \rangle + \langle \mathbf{c}_2, \mathbf{s} \rangle = \left\lfloor \frac{q}{p} \right\rfloor \cdot (m_1 + m_2) + E_1 + E_2$$

so the error terms simply add.

Multiplication. Now, consider the case of multiplication. First, we consider the error δ_1 introduced by rounding. To facilitate the analysis, define

$$\mathbf{c}' \triangleq \left\lfloor \frac{p}{q}\left(\mathbf{c}_1 \otimes \mathbf{c}_2\right) \right\rceil - \frac{p}{q}\left(\mathbf{c}_1 \otimes \mathbf{c}_2\right).$$

The error δ_1 introduced by rounding is then

$$\delta_1 \triangleq \left\langle \left\lfloor \frac{p}{q} \cdot (\mathbf{c}_1 \otimes \mathbf{c}_2) \right\rceil, \mathbf{s} \otimes \mathbf{s} \right\rangle - \left\langle \frac{p}{q}\left(\mathbf{c}_1 \otimes \mathbf{c}_2\right), \mathbf{s} \otimes \mathbf{s} \right\rangle = \left\langle \mathbf{c}', \mathbf{s} \otimes \mathbf{s} \right\rangle.$$

By construction, we see that $\|\mathbf{c}'\|_\infty \leq \frac{1}{2}$ and so $\|\mathbf{c}'\|_\infty^{\text{can}} \leq \frac{\varphi(m)}{2}$. Next, noting that $\tilde{\mathbf{s}} \sim \mathcal{DG}_m(h)$, we have that $\|\tilde{\mathbf{s}}\|_\infty^{\text{can}} \leq 6\sqrt{h}$ and $\left\|\tilde{\mathbf{s}}^2\right\|_\infty^{\text{can}} \leq 16h$ and so

$$\|\mathbf{s} \otimes \mathbf{s}\|_1^{\text{can}} \leq \|1\|_\infty^{\text{can}} + 2\,\|\tilde{\mathbf{s}}\|_\infty^{\text{can}} + \left\|\tilde{\mathbf{s}}^2\right\|_\infty^{\text{can}} = 1 + 12\sqrt{h} + 16h$$

Thus, we can bound $\|\delta_1\|_\infty^{\text{can}}$ with

$$\|\delta_1\|_\infty^{\text{can}} \leq \left\|\mathbf{c}'\right\|_\infty^{\text{can}} \|\mathbf{s} \otimes \mathbf{s}\|_1^{\text{can}} \leq \frac{1}{2}\varphi(m)\left(1 + 12\sqrt{h} + 16h\right).$$

Now, let $I_1, I_2 \in R$ be such that

$$\langle \mathbf{c}_1, \mathbf{s} \rangle = \left\lfloor \frac{q}{p} \right\rfloor \cdot m_1 + E_1 + q \cdot I_1$$
$$\langle \mathbf{c}_2, \mathbf{s} \rangle = \left\lfloor \frac{q}{p} \right\rfloor \cdot m_2 + E_2 + q \cdot I_2.$$

We begin by bounding the canonical norm of I_1 (clearly, the same bound holds for I_2):

$$\begin{aligned}
\|I_1\|_\infty^{\text{can}} &= \frac{\left\| \langle \mathbf{c}_1, \mathbf{s} \rangle - \left\lfloor \frac{q}{p} \right\rfloor \cdot m_1 - E_1 \right\|_\infty^{\text{can}}}{q} \\
&\leq \frac{\|\langle \mathbf{c}_1, \mathbf{s} \rangle\|_\infty^{\text{can}}}{q} + \frac{\left\| \left\lfloor \frac{q}{p} \right\rfloor m_1 \right\|_\infty^{\text{can}}}{q} + \frac{\|E_1\|_\infty^{\text{can}}}{q} \\
&\leq \frac{\|\mathbf{c}_1\|_\infty^{\text{can}} \|\mathbf{s}\|_1^{\text{can}}}{q} + \varphi(m) \left\| \frac{\left\lfloor \frac{q}{p} \right\rfloor m_1}{q} \right\|_\infty + \varphi(m) \left\| \frac{E_1}{q} \right\|_\infty \\
&\leq \frac{\varphi(m) \|c_1\|_\infty (1 + 6\sqrt{h})}{q} + \varphi(m) + \varphi(m) \\
&\leq \frac{q(1 + 6\sqrt{h})}{q} \varphi(m) + 2\varphi(m) \\
&= (6\sqrt{h} + 3)\varphi(m).
\end{aligned}$$

Then,

$$\begin{aligned}
\langle \mathbf{c}_{\text{mult}}, \mathbf{s} \otimes \mathbf{s} \rangle - \delta_1 &= \frac{p}{q} \left(\left\lfloor \frac{q}{p} \right\rfloor \cdot m_1 + E_1 + q \cdot I_1 \right) \left(\left\lfloor \frac{q}{p} \right\rfloor \cdot m_2 + E_2 + q \cdot I_2 \right) \\
&= \left\lfloor \frac{q}{p} \right\rfloor m_1 m_2 + \delta_2 + q \left(m_1 I_2 + m_2 I_1 + p I_1 I_2 \right)
\end{aligned}$$

where δ_2 is defined as

$$\delta_2 \triangleq (pE_2 - rm_2)I_1 + (pE_1 - rm_1)I_2 + \frac{q-r}{q}(E_1 m_2 + E_2 m_1) + \frac{p}{q} E_1 E_2 - \frac{r m_1 m_2}{pq}(q - r),$$

and $r = [q]_p$. Let $E = \max\{\|E_1\|_\infty^{\text{can}}, \|E_2\|_\infty^{\text{can}}\}$. Thus, we can bound $\|\delta_2\|_\infty^{\text{can}}$ by

$$\begin{aligned}
\|\delta_2\|_\infty^{\text{can}} &\leq 2(pE + p^2\varphi(m))(6\sqrt{h} + 3)\varphi(m) + 2p\varphi(m)E + \frac{pE^2}{q} + p^2\varphi^2(m) \\
&\leq p \left[2\varphi(m)(6\sqrt{h} + 4) + 1 \right] E + p^2 \varphi^2(m) \left[2(6\sqrt{h} + 3) + 1 \right].
\end{aligned}$$

Summarizing results, we have

$$\begin{aligned}
\langle \mathbf{c}_{\text{mult}}, \mathbf{s} \otimes \mathbf{s} \rangle &= \left\lfloor \frac{q}{p} \right\rfloor m_1 m_2 + \delta_1 + \delta_2 \pmod q \\
&\leq \left\lfloor \frac{q}{p} \right\rfloor m_1 m_2 + p \left[2\varphi(m)(6\sqrt{h} + 4) + 1 \right] E \\
&\quad + \left[\frac{1}{2}\varphi(m) \left(1 + 12\sqrt{h} + 16h \right) + p^2 \varphi^2(m) \left[2(6\sqrt{h} + 3) + 1 \right] \right] \pmod q
\end{aligned}$$

where E is a bound on the initial amount of noise in $\mathbf{c}_1$ or $\mathbf{c}_2$.

Scalar Multiplication. When we multiply a ciphertext $\mathbf{c}$ (with error E) by a scalar $\alpha \in R_p$, we have,

$$\langle \mathbf{c}_{\text{scale}}, \mathbf{s} \rangle = \langle \alpha \mathbf{c}, \mathbf{s} \rangle = \alpha \langle \mathbf{c}, \mathbf{s} \rangle = \left\lfloor \frac{q}{p} \right\rfloor \alpha m + \alpha E.$$

Since $\alpha \in R_p$, we have that $\|\alpha\|_\infty^{\text{can}} \leq \varphi(m) \|\alpha\|_\infty \leq \varphi(m) \cdot p$. In general, the error increases by a multiple of $\varphi(m) \cdot p$. Naturally, we can get a better bound based on the value of α. For example, if $\alpha = k \in \mathbb{Z}/p\mathbb{Z}$ (i.e., a constant polynomial), then the noise will only increase by a factor of $|k|$.

Automorphism. As noted in [GHS12a, GHS12b], the canonical embeddings of a polynomial $f \in R_q$ and $\kappa_i(f)$ for $i \in (\mathbb{Z}/m\mathbb{Z})^*$ are just permutations of each other. In other words, applying an automorphism does not change the canonical embedding norm of the underlying polynomial.

5.3 Security Analysis and Parameter Selection

Following the analysis presented in [GHS12b], to ensure a time/advantage ratio of at least 2^k, we need to set the size $N = \varphi(m)$ of our RLWE instance to be at least

$$\varphi(m) = N \geq \frac{\log(q/\sigma)(k + 110)}{7.2}. \tag{7}$$

Targeting a 128-bit security level, we require $\varphi(m) \geq \log(q/\sigma) \cdot 33.1$. Using these expressions, we can derive a bound on N. Following the conventions in [GHS12b], we take $\sigma = 3.2$ and $h = 64$. Recall that to achieve 128-bit precision, we perform the computation with respect to primes $p_1, \ldots, p_n$ such that $\prod_{i=1}^n p_i \geq 2^{128}$. As noted in Section 4.3, we take our primes $p_1, \ldots, p_n$ to be safe primes with $m_i = p_i - 1$ for all $i = 1, \ldots, n$.

Linear Regression. We proceed to compute the parameters necessary to run linear regression for a given dimensionality z. Let ξ denote the number of ciphertext blocks in the data (recall that each block contains many packed plaintext elements). Substituting $\sigma = 3.2$, $h = 64$, $b = 2^{24}$, and $p_i = 2\varphi(m_i) + 3 = 2N + 3$ into (6),

$$E_{\text{clean}} \leq 8\sqrt{2}\sigma\varphi(m) + 6\sigma p\sqrt{\varphi(m)} + 16\sigma p\sqrt{h \cdot \varphi(m)} \approx 1287\sqrt{N} + 37N + 858N^{3/2} \leq 2^{10}N^{3/2},$$

where the last inequality holds if $N \geq 9$. First, we consider the noise added at each level of the computation. First, we perform a multiply, followed by a sum of ξ elements or blocks. This introduces an error E_1 bounded by

$$\begin{aligned} E_1 &= \xi \left(p \left[2N(6\sqrt{h} + 4) + 1 \right] E + p^2 N^2 \left[2(6\sqrt{h} + 3) + 1 \right] \right) \\ &\leq \xi \left[E \left(2N + 3 \right) \left(2^{6.8} N \right) + 2^{6.7} (2N + 3)^2 N^2 \right] \\ &\leq \xi \left[E \left(2^{8.4} N + 2^{7.9} N^2 \right) + 2^{9.9} N^2 + 2^{10.3} N^3 + 2^{8.7} N^4 \right]. \end{aligned}$$

Next, we round the result of the multiplication and apply a key switch from the tensored key to the base key. This introduces a noise E_2 bounded by

$$E_2 = \frac{1}{2} N \left(1 + 12\sqrt{h} + 16h \right) + 6\sigma b(4 + 1) \left\lceil \log_b q \right\rceil N^{3/2} \leq 2^{9.2} N + 2^{30.6} \left\lceil \log_b q \right\rceil N^{3/2}.$$

To simplify the bound, we restrict our attention to solutions where $\lceil \log_b q \rceil \leq 128$. From an implementation perspective, we are interested in choosing the smallest parameters N and q such that the security and

correctness bounds hold. Thus, imposing some reasonable upper bounds on the size of the parameters is not wholly unjustified. In this case, we are requiring that $q < 2^{3072}$, which is easily satisfied by our solutions. With this in mind, the total amount of noise introduced in a single level of computation is bounded by

$$E_1 + E_2 = \xi \left[E \left(2^{8.4} N + 2^{7.9} N^2 \right) + 2^{9.9} N^2 + 2^{10.3} N^3 + 2^{8.7} N^4 \right] + 2^{9.2} N + 2^{37.6} N^{3/2}.$$

We can simplify this bound by placing some small restrictions on N. For $N \geq 20$, we have that $2^8 N^2 \geq 2^{8.4} N + 2^{7.9} N^2$. Similarly, for $\xi \geq 1$ and $2^{23} \geq N \geq 450$, we have

$$\begin{aligned} \xi \left[2^{9.9} N^2 + 2^{10.3} N^3 + 2^{8.7} N^4 \right] + 2^{9.2} N + 2^{29.6} N^{3/2} &\leq \xi \left[2^{9.9} N^2 + 2^{10.3} N^3 + 2^{8.7} N^4 + 2^{9.2} N + 2^{37.6} N^{3/2} \right] \\ &\leq \xi \left[2^{20} N^{7/2} \right] = \xi \left[2^{10} N^2 E_{\text{clean}} \right]. \end{aligned}$$

Finally, we note that E_{clean} is a lower bound on the noise (no operation can decrease the noise) and so we have

$$E_1 + E_2 \leq \xi \left[2^8 N^2 E + 2^2 N^2 E_{\text{clean}} \right] \leq \xi N^2 E \left[2^8 + 2^{10} \right] = 1280 N^2 \xi E.$$

Thus, performing one multiplication followed by ξ additions and a key switch increases the error by at most a factor of $1280 N^2 \xi$. Now, when we compute $A^T A$ and $A^T y$, we also need to evaluate the sum of the elements within the block. As described in Section 4.2, this step requires a series of automorphisms and summations. Assuming a batch size of $2^{\lfloor \log N \rfloor}$, we need to perform a series of $\lfloor \log N \rfloor$ automorphisms. In order to sum up the results, we perform a key switch after each automorphism, which introduces an error E_{auto} given by

$$\begin{aligned} E_{\text{auto}} = \frac{1}{2} N \left(1 + 12\sqrt{h} + 16h \right) + 6\sigma b (2+1) \left\lceil \log_b q \right\rceil N^{3/2} &\leq 2^{9.2} N + 2^{36.9} N^{3/2} \\ &\leq 2^{37} N^{3/2} \end{aligned}$$

To evaluate the sum, we note that the additions and automorphisms are nested. Consequently, each sum doubles the error. Given $\lfloor \log N \rfloor$ automorphisms, the error increases by a multiple of $2^{\lfloor \log N \rfloor} \leq N$. Letting E denote the error in the ciphertext block at the beginning of this operation, computing the sum increases the error by at most $N \left(E + 2^{37} N^{3/2} \right)$.

Thus, to evaluate the linear regression circuit over data with dimension z, we first form $A^T A$ and $A^T y$. Given a clean encryption with noise E_{clean}, these products will have an error of at most

$$N \left(1280 N^2 \xi E_{\text{clean}} + 2^{37} N^{3/2} \right).$$

Evaluating the remainder of the circuit requires $\max \{1, z-1\}$ additional multiplications, which yields an error of

$$\begin{aligned} N \left(1280 N^2 E_{\text{clean}} + 2^{37} N^{3/2} \right) \left(1280 N^2 \xi \right)^{\max\{1, z-1\}} &\leq N^{3/2} \left(2^{20.4} N^2 + 2^{37} \right) \left(1280 N^2 \xi \right)^{\max\{1, z-1\}} \\ &\leq N^{3/2} \left(2^{21} N^2 \right) \left(1280 N^2 \xi \right)^{\max\{1, z-1\}} \\ &\leq 2^{21} N^{7/2} \left(1280 N^2 \xi \right)^{\max\{1, z-1\}} \end{aligned}$$

where the second line follows for $N \geq 440$. Finally, as described in Section 4.2, we need to introduce randomness into the second through last plaintext slots. For each ciphertext $\mathbf{c}$ that we need to introduce randomness, we construct an encryption $\mathbf{c}_{\text{rand}}$ of a batched plaintext with uniformly random values in the second through last slot and zero in the first slot. We then compute $\mathbf{c} + \mathbf{c}_{\text{rand}}$. Upon decryption, the value

in the first slot will be preserved while the values in the remaining slots will be uniformly random. Since $\mathbf{c}_{\text{rand}}$ is a fresh encryption of a polynomial in R_p , this addition introduces an additive error of E_{clean}. Thus, after performing the necessary computations, the error in the resulting ciphertexts is bounded by

$$\begin{aligned} 2^{21}N^{7/2}\left(1280N^2\xi\right)^{\max\{1,z-1\}} + E_{\text{clean}} &= 2^{21}N^{7/2}\left(1280N^2\xi\right)^{\max\{1,z-1\}} + 2^{10}N^{3/2} \\ &\leq 2^{21.1}N^{7/2}\left(1280N^2\xi\right)^{\max\{1,z-1\}} \end{aligned}$$

For decryption to succeed, we require that

$$2^{21.1}N^{7/2}\left(1280N^2\xi\right)^{\max\{1,z-1\}} \leq \frac{\lfloor q/2 \rfloor}{c_m \cdot p} \leq \frac{q}{2 \cdot c_m \cdot (2N+3)}.$$

This yields

$$q \geq 2^{22.1}c_m N^{7/2}(1280N^2\xi)^{\max\{1,z-1\}}(2N+3)$$

For our choices of m, $c_m \leq 2$. Furthermore, $3N \geq 2N+3$ for $N \geq 3$. Thus, it is sufficient to satisfy

$$q \geq 2^{24.7}N^{9/2}(1280N^2\xi)^{\max\{1,z-1\}} \tag{8}$$

We can now use Equations (7) and (8) to obtain values of q and N that ensure security and correctness. Approximate values that satisfy these conditions for different values of z and with $\xi = 256$ are presented in Table 1.

Mean and Covariance. We also consider the parameters needed to estimate data-covariance matrices. Again, let z denote the dimension of the data. To compute the covariance matrix, we evaluate Equation 2. Clearly, computing nX^TX (two levels of multiplication with an automorph-sum operation) dominates the cost of the evaluation. Now, we note that computing nX^TX is analogous to evaluating the linear regression circuit with $z = 2$. Thus, we simply cite the result from above:

$$q \geq 2 \cdot 2^{24.7}N^{9/2}1280N^2\xi \approx 2^{36.1}N^{13/2}\xi.$$

Note that the extra factor of 2 is due to the extra addition that we perform. For $\xi = 256$ and ensuring 128-bit security, we require $N \geq 3977$.

6 Experiments and Conclusion

In this section, we describe the experiments we conducted using the above described scheme. Similar to [GHS12b], our implementation was based on the NTL C++ library. For all experiments, we used a machine with a 24-core AMD Opteron processor running at 2.1 GHz and 512 KB of cache and 96 GB of available memory. Note that because NTL is not thread-safe, we ran all of our tests in a single-threaded environment. Furthermore, the peak memory usage throughout our tests was on the order of 20-40 GB for the largest experiments.

As described in in Section 4.2, to achieve 128-bit precision in the output, we perform the computation with respect to many small primes and then apply CRT to obtain the final output. In our experiments, we first determine the necessary parameters needed for security and correctness according to the analysis in Section 5.3. Once we have determined the minimum values for m and q, we compute the smallest $p_1, \ldots, p_k$ such that $p = \prod_{i=1}^{k} p_i > 2^{128}$ and $m_1 = p_1 - 1, \ldots, m_k = p_k - 1$ satisfy the necessary bounds. For our experiments, we measured the time necessary to perform the computation with respect to one of

Dimension of Data	$N = \varphi(m)$	$\log q$	p_1	p_n	Batch Size
1	3933	120.7	8423†	9743	4096
2	3933	120.7	8423†	9743	4096
3	5461	166.9	11003	12263	4096
4	7013	213.9	14087	15083	4096
5	8583	261.4	17327	18743	8192

† We used slightly larger primes than necessary in order to take advantage of increased batching. In particular, we took $N \geq 4096$ in these experiments.

Table 1: Parameters for linear regression, computed to satisfy security (7) and correctness (8) in the RLWE-based scale-invariant FHE scheme . The listed values for N and $\log q$ are the smallest such values that are correct and secure. The listed values for p_1 and p_n indicate the smallest and largest primes, respectively, we use in our implementation to obtain 128 bit precision in the computation. Note that for each prime, we perform the computation with respect to a different $N = \varphi(m)$. Thus, we make small adjustments to the value of q so Equations (7) and (8) still hold. The values here are computed taking $\xi = 256$.

	Time for each Step (minutes)					
Dataset Size	Key Generation	Batching	Encryption	Regression	Decryption	Total
1024	1.74	0.03	0.06	5.01	0.18	7.02
4096	1.73	0.13	0.06	5.01	0.18	7.12
8192	1.74	0.25	0.12	5.05	0.18	7.35
16384	1.74	0.51	0.24	5.04	0.18	7.71
65536	1.73	2.05	0.96	5.66	0.19	10.60
262144	2.27	8.10	4.30	9.38	0.19	24.23
1048576	2.13	32.96	16.20	18.39	0.18	69.87
4194304	2.15	132.27	64.77	56.71	0.19	256.08

Table 2: Linear regression run time as a function of dataset size. All timing experiments were performed over 2-dimensional data with plaintext modulus $p = 9743$. The batch size across all experiments was 4096.

the primes in the chain (generally, the largest or close to largest one); thus, the reported times should serve as an approximate upper bound on the time needed to perform one such iteration of this procedure. In practice, we can perform all k computations in parallel since the individual computations are independent. For small k, this is certainly feasible using multicore or distributed systems.

Linear Regression. We consider two sets of experiments. In the first experiment, we measure the system's run time as we increase the number of data points in our regression example and in the second, we measure the run time as we increase the dimensionality of the data. In particular, we consider a maximum data dimension of 5 as well as datasets consisting of around four million (2^{22}) elements. These results are summarized in Tables 2 and 3.

Mean and Covariance. Similarly, we measure the system's run time as we increase the number of data points in the dataset as well as the dimensionality of the data. In this case, we consider a maximum dimension of 24 and datasets consisting of over a million (2^{20}) elements. We present results from these experiments in Tables 4 and 5.

By batching multiple plaintext values into a single ciphertext block, we are able to significantly reduce the number of operations needed to compute inner products and correspondingly, matrix products. In doing so, it becomes possible to significantly scale up the number of data elements we can feasibly process

		Time for each Step (minutes)					
Dataset Size	Dimension	Key Generation	Batching	Encryption	Regression	Decryption	Total
4096	1	1.83	0.09	0.04	1.57	0.12	3.65
	2	1.74	0.13	0.06	5.00	0.18	7.10
	3	3.58	0.22	0.12	18.79	0.34	23.05
	4	5.42	0.33	0.18	54.79	0.55	61.27
	5	16.48	0.50	0.49	412.61	1.66	431.75
65536	1	1.73	1.35	0.64	1.76	0.12	5.59
	2	1.74	2.03	0.96	5.64	0.18	10.55
	3	3.51	3.43	1.90	20.25	0.35	29.44
	4	5.86	5.28	3.16	64.02	0.56	78.88
	5	16.08	7.91	4.06	410.98	1.67	440.71
262144	1	2.13	5.40	2.70	2.93	0.12	13.28
	2	2.22	8.13	4.21	9.14	0.19	23.89
	3	3.43	13.68	7.52	25.99	0.35	50.98
	4	5.96	20.96	12.85	76.27	0.57	116.60
	5	16.66	32.20	17.10	449.60	1.68	517.24

Table 3: Linear regression run time as a function of data dimension. The plaintext modulus and batch size for each dimension are as given in Table 1. Specifically, we performed the experiment with respect to the largest plaintext modulus p_n needed to achieve 128-bit precision.

		Time for each Step (minutes)					
Dimension	Dataset Size	Key Generation	Batching	Encryption	Computation	Decryption	Total
4	4096	1.74	0.17	0.10	12.94	0.89	15.84
	8192	1.87	0.34	0.21	14.01	0.92	17.35
	16384	1.81	0.68	0.42	13.66	0.90	17.48
	65536	1.75	2.75	1.62	14.53	0.92	21.57
	262144	2.17	11.11	6.81	21.28	0.92	42.30
	1048576	2.17	44.47	27.09	40.89	0.91	115.53
8	4096	1.78	0.34	0.18	42.20	2.83	47.33
	8192	1.75	0.68	0.35	41.88	2.88	47.54
	16384	1.75	1.36	0.72	42.63	2.87	49.34
	65536	1.76	5.44	2.90	46.90	2.83	59.83
	262144	2.14	21.77	12.07	70.09	2.93	109.01
	1048576	2.22	87.41	49.24	142.80	2.91	284.57

Table 4: Mean and covariance computation run time as a function of the dataset size. In all experiments, the plaintext modulus was taken to be $p = 9743$ and the batch size was taken to be 4096.

	Time for each Step (minutes)					
Dimension	Key Generation	Batching	Encryption	Computation	Decryption	Total
1	1.85	0.04	0.04	1.89	0.15	3.97
2	1.77	0.09	0.06	4.62	0.34	6.87
4	1.84	0.17	0.10	13.57	0.90	16.58
8	1.77	0.34	0.18	42.24	2.88	47.40
12	1.74	0.51	0.26	85.33	5.90	93.75
16	1.82	0.68	0.35	152.22	10.23	165.30
20	2.15	0.86	0.44	254.24	16.35	274.04
24	2.27	1.02	0.56	377.81	24.26	405.92

Table 5: Mean and covariance computation run time as a function of data dimension. All experiments were conducted over datasets with 4096 data points. The plaintext modulus was fixed at $p = 9743$ and the batch size at 4096.

Dimension	Batch Size	Batching Time Per Element (ms)	Encryption Time Per Block (s)
1	4096	1.24	2.53
2	4096	1.86	3.94
3	4096	3.13	7.05
4	8192	4.80	12.05
5	8192	7.37	32.06

Table 6: Per block and per element computation time for batching and encryption. Numbers based upon parameters for running regression over 2^{18} data points. Note that we do not adjust for the dimensionality of the data. For instance, one "block" of 5-dimensional data is equivalent to 5 blocks of 1-dimensional data and correspondingly for elements.

using a leveled homomorphic encryption scheme. Specifically, we are able to perform 2-dimensional linear regression over four million elements in just over five hours. Furthermore, we note that most of the time in this particular case is dominated by the batching and encryption time. In an actual implementation of this scheme, we would have a large number of data providers, each supplying a few batches of data to the central server. In the case of batching, the actual quantity of interest is the amount of time batching takes per element. Although it is most efficient if each data provider submits a complete batch of data rather than multiple incomplete batches, this is by no means necessary. In the case of encryption, each provider always encrypts an integral number of blocks, so the metric of interest is the time needed to encrypt a single block of data. Observe that this is independent of the number of data elements packed within the block. Table 6 gives the cost of batching on a per-element basis and the cost of encryption on a per-block basis. We can easily see that even for five-dimensional data, the work needed to be done by each data provider is relatively low: on the order of a few minutes to batch and encrypt a sizable quantity of data.

As expected, our regression scheme only works well for relatively low dimension data. This is due directly to the high cost of using Cramer's rule for matrix inversion. In particular, computing the adjugate and determinant of the matrix scales as $O(d!)$ where d is the dimension of the data. Thus, this particular method of matrix inversion is not well-suited for higher dimension data. Using alternative methods for matrix inversion will be essential to a system that works for higher dimensional data.

In the case of mean and covariance computation, the cost is dominated by the cost of computing the covariance matrix. Because the complexity of the computation is quadratic in both the dimension as well as the size of the dataset, it is possible to scale these computations up more substantially. In particular, we are able to compute the covariance matrix of 4096 elements, each with dimension 24, in just under 17 hours. We also note that computing the matrix products can easily be parallelized. As such, by leveraging parallel programming, we can further scale up the covariance computation.

For our final set of experiments, we evaluated how the running time scales as we vary the security level. In particular, we consider the running time of linear regression assuming as low as 80-bit security to as high as 1024-bit security. Increasing security directly translates to using larger RLWE instances, and thus, larger parameters. Table 7 summarizes our experiments with running linear regression over 2 and 3-dimensional datasets with 16384 elements. It is interesting to note that the jumps in run time occur when the batch size increases; this is indicative of the fact that the series of key-switches we perform to sum up the element within blocks dominates when the number of block multiplications is relatively small. We also note that the rate of increase in the size $N = \varphi(m)$ of the RLWE-instances as well as the modulus p is effectively linear in the security parameter. For low dimensional cases with just a few blocks of data, the run time scales roughly linear as well. For higher dimensional problems with more blocks, there will be many more multiplications that need to be performed, so the run time would be expected to scale super-linearly, most likely quadratically, with the security parameter. Nonetheless, it is reassuring to observe that even with 1024-bit security, it is still possible to regress on more than 10000 data points in an hour or two.

Dimension	Security Level	$\log q$	p	Batch Size	Regression Time (min)	Total Time (min)
2	80	115	6983	2048	2.63	4.17
	128	116	8423	4096	4.93	7.46
	256	120	12899	4096	5.95	9.11
	512	124	21767	8192	15.30	23.07
	1024	130	40823	16384	38.14	57.66
3	80	154	9467	4096	15.59	19.84
	128	155	10883	4096	18.32	23.32
	256	161	17903	8192	40.05	50.52
	512	166	28607	8192	48.44	61.73
	1024	174	54959	16384	123.68	159.97

Table 7: Timing tests as a function of the security parameter. In all experiments, we performed linear regression on datasets consisting of 16384 elements. The parameters were chosen based on our choice of dataset size.

In this paper, we have demonstrated a viable method of using fully homomorphic encryption for large scale statistical analysis. We note that such a scheme is particularly well-suited in scenarios where we have multiple sources of data (i.e., many sensors in the field) that need to be aggregated and analyzed. In this massively distributed setting, the cost of batching and encryption on a per-element and per-block basis is relatively inexpensive. On the server-side, we have demonstrated that by taking advantage of batching and using a CRT-based message encoding, it becomes feasible to process both significantly larger datasets and higher dimensional data, particularly in comparison to prior work. In turn, we have conducted experiments involving datasets with more than four million elements as well as data with more than 20 dimensions.

References

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

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

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. Cryptology ePrint Archive, Report 2012/078, 2012. http://eprint.iacr.org/2012/078.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. Cryptology ePrint Archive, Report 2011/344, 2011. http://eprint.iacr.org/2011/344.

[DPSZ11] I. Damgard, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. http://eprint.iacr.org/2011/535.

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Crypto*, pages 10–18, 1985.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Eurocrypt*, pages 465–482, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Report 2012/099, 2012. http://eprint.iacr.org/2012/099.

[GLN12] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. Cryptology ePrint Archive, Report 2012/323, 2012. http://eprint.iacr.org/2012/323.

[HFN11] Robert Hall, Stephen Fienberg, and Yuval Nardi. Secure multiparty linear regression based on homomorphic encryption. In *Journal of Official Statistics*, pages 669–691, 2011.

[LNV11] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? Cryptology ePrint Archive, Report 2011/405, 2011. http://eprint.iacr.org/2011/405.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Eurocrypt*, 2010.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, pages 223–238. Springer-Verlag, 1999.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[SV11] N.P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011. http://eprint.iacr.org/2011/133.