Scalable Secure Computation of Statistical Functions with Applications to $k$-Nearest Neighbors

Hayim Shaul, Dan Feldman, Daniela Rus

June 29, 2018

Abstract

Given a set $S$ of $n$ $d$-dimensional points, the $k$-nearest neighbors (KNN) is the problem of quickly finding $k$ points in $S$ that are nearest to a query point $q$. The $k$-nearest neighbors problem has applications in machine learning for classifications and regression and also in searching. The secure version of KNN where either $q$ or $S$ are encrypted, has applications such as providing services over sensitive (such as medical or localization) data.

In this work we present the first scalable and efficient algorithm for solving KNN with Fully Homomorphic Encryption (FHE) that is realized by a polynomial whose degree is independent of $n$, the number of points. We implemented our algorithm in an open source library based on HELib implementation for the Brakerski-Gentry-Vaikuntanths FHE scheme, and ran experiments on MIT’s OpenStack cloud. Our experiments show that given a query point $q$, we can find the set of 20 nearest points out of more than a thousand points in less than an hour.

Our result introduces a statistical coreset, which is a data summarization technique that allows statistical functions, such as moments, to be efficiently and scalably computed. As a central tool, we design a new coin toss technique which we use to build the coreset. This coin toss technique and computation of statistical functions may be of independent interest.

1 Introduction

The use of cloud services is increasing dramatically. Indeed, using a service located in the cloud has its advantages over downloading and installing the service locally. Consider, for example, (i) a map that can be queried for a site (such as a restaurant or a hotel) near a certain location; (ii) a diagnostics tool for skin cancer, where a user uploads an image of her skin; or (iii) a recommender system based on previous activities of similar profiles. Even if the databases are small enough to be installed on the user’s device, a cloud service can constantly updates its databases and algorithms to have more accurate and more relevant results, something that is hard for a user to do. The latter two examples, being examples of machine learning indeed require updates to the databases as more data is collected from willing users, and also updates to the algorithms as they are improved. In the first example as well, updates are required to the database as sites are added or removed. One big disadvantage of a cloud service is the loss of privacy for the user. The user is expected to reveal its data to be able to use the service. In some cases this data may be something the user does not wish to reveal. For example, the output of a recommender system can also be used to perform differential pricing on the expense of the user. In other cases, it is the interest of the service not to know user’s information as it may impose extra regulation on the service. For example, having access to user medical data may require it to comply with HIPAA regulations.

Multi-party computation (MPC) is used to let two or more parties compute a certain output from their inputs without disclosing any information (except maybe for the output) to the other party.

**Fully Homomorphic Encryption (FHE)** is a secret-key-public-key encryption scheme that also supports + and $\times$ operations on the ciphertext. To encrypt a message $m$, the public key is used $m' = \ldots$
Encrypt(m, pub), and decryption is possible only with the secret key $m = Decrypt(m', sec)$. In addition we have, $Decrypt(Encrypt(x, pub) + Encrypt(y, pub), sec) = x + y$ and $Decrypt(Encrypt(x, pub) \times Encrypt(y, pub), sec) = x \times y$. This suggests that any polynomial $P$ (and in fact any algorithm) can be computed on encrypted messages $(x_1, x_2, \ldots)$ by applying $+$ and $\times$ operations on encrypted values to get $P(x_1, x_2, \ldots)$. In the context of multi-party computation (MPC), FHE can be used to construct a single-round protocol where one party (the cloud) applies a polynomial on user’s data, such that the cloud does not learn anything on the user’s data. The result of applying a polynomial $P$ on encrypted input $Encrypt(x)$ is an encrypted value $Encrypt(P(x))$ that can only be decrypted by the user with her secret key. Except for $P(x)$, the user does not learn anything about $P$.

The implementation of FHE schemes usually involves encoding the data $m$ using some transformation over a finite ring while adding some noise. Removing the noise and reversing the transformation is possible only with the secret key. As operations are performed on ciphertexts the noise grows. The user needs to make sure the generated secret key will be able to remove the maximal amount of noise accumulated during the computation of $P(x)$. There are two techniques to reduce noise: modulo switching and bootstrapping. The latter is not CPU efficient, as it involves the processes of decrypting a ciphertext under FHE. Modulo switching, on the other hand, is very efficient to use, however, the number of times it can be used is set at the time the key pair is generated. With FHE schemes known to date, as key-pair are generated to support more modulo-switching, they grow in size, take more time to generate and render operations performed on ciphertext to have more overhead.

The main challenge is when designing an algorithm for FHE is to make it compatible with keys generated with the least amount of modulo switching. Since the noise accumulated in a $\times$ operation is significantly larger than the noise accumulated in a $+$ operation, the latter is usually ignored and the challenge becomes to design an algorithm whose equivalent polynomial has a low degree. Although theoretically any algorithm can be expressed as a polynomial of its inputs, in practice, the naive polynomials are rarely practical. Hereinafter, computations are assumed to be compatible to the FHE model (arithmetic circuit model), where we are restricted to use $+$ and $\times$ over some finite ring.

Definition 1.1 (Polynomial’s size and depth) A function $f : \mathbb{Z}_p^s \rightarrow \mathbb{Z}_p^t$ is a polynomial over a finite ring $\mathbb{Z}_p$ if it is a multivariate polynomial that maps a vector $x$ of $s$ ring elements in $\mathbb{Z}_p$, to a vector $y = f(x)$ of $t$ ring elements in $\mathbb{Z}_p$, which is the result of the polynomial $f$ on the input $x$. The complexity of the computation is measured by the degree of the polynomial $f$, denoted by deg$(f)$, and the number of multiplications needed to evaluate it, that is denoted by size$(f)$.

For a general polynomial, $P$, with $\deg(P) = n$, we have $\text{size}(P) = \sqrt{n}$. See [31] for details. However, for some polynomials, not necessarily sparse, smaller bounds exist. For example, over $\mathbb{Z}_p$, where $p$ is prime, the polynomial $\sum \binom{p-1}{i} x^i = (x+1)^{p-1} - 1$, whose value is 0 for $x = p - 1$ and 1 otherwise, has size $O(\log p)$. Motivated by FHE problems, we measure the depth and size of the polynomials in our solution. We focus on reducing their degree and their size.

Comparing encrypted numbers. Unlike comparison model, implementing a comparison in FHE is not trivial. Simply put, FHE does not provide a comparison operator. Having the ability to compare two encrypted numbers and getting an unencrypted result would essentially break the security of the encryption as it leaks information on the ciphertext. However, for two encrypted numbers $[x]$ and $[y]$ it is possible to construct a polynomial $P(x, y)$ whose value is encrypted and equals 1 if $x < y$ and 0 otherwise.

$k$-nearest neighbors (KNN) is a problem where given a finite set $S \subset \mathbb{R}^d$, with $|S| = n$ and a point $q \in \mathbb{R}^d$ we are looking for a subset $S' \subset S$, of the $k$ nearest points to $q$, i.e. $|S'| = k$ and $\max_{x \in S'} \|x - q\| \leq \min_{x \in S \setminus S'} \|x - q\|$. In machine learning the KNN algorithm is used as a classifier, when each point $x \in S$ is assigned a class and the class of $q$ is taken to be the majority of the classes in $S'$. Specifically, KNN can address the examples mentioned above. Setting $S \subset \mathbb{R}^2$ to be a set of “sites” and $q \in \mathbb{R}^2$ to be users location, KNN is the subset of $k$ nearest sites to $q$. Setting $S \subset \mathbb{R}^2$ to be a training set for skin cancer, labeled as
“benign” or “malignant”, and \( q \in \mathbb{R}^d \) be a sample to be classified, KNN is a subset of \( k \) nearest training sample to \( q \) from which \( q \) can be classified as “benign” or “malignant”. Lastly, setting \( S \subset \mathbb{R}^d \) to be profiles, each paired with a recommendation, and \( q \in \mathbb{R}^d \) to be a user’s profile, KNN is a subset of \( k \) most similar profiles to \( q \) which suggests \( k \) recommendations to the user.

Despite its simplicity, \( k \)-nearest neighbors is a hard problem to solve in FHE. Informally, a polynomial whose value is the index of one of the \( k \) nearest neighbors should somehow consider all data points in \( S \). Naively, projecting from a comparison-model algorithm, one would expect this to be a polynomial of degree \( O(n) \). Making it harder, is the fact that in the FHE model an equivalent of the \(< \) operator needs to be implemented from + and \( \times \) operators which naively results in a polynomial with degree \( O(n \cdot \deg(<)) \), where \( \deg(<) \) is the degree of the polynomial evaluating the \(< \) operation.

**Statistical functions** such as the cumulative distribution function (CDF) can be used to solve the KNN problem. Specifically, for a random variable \( d = \| q - s_i \| \), where \( i \) is drawn uniformly from \( \{1, \ldots, n\} \), and \( CDF_d \) being its CDF then by definition we have, \( |\{ s_i \in S \mid \| q - s_i \| < T \}| = k \), where \( T = CDF_d^{-1}(k/n) \).

A closely related set of functions are **moments**. Given a data set \( D = \{d_1, \ldots, d_n\} \), the \( e \)-th moment is defined as \( \mu_e(D) = \frac{1}{n} \sum_i d_i^e \). The first order moment, with \( e = 1 \), is the average of \( D \). The variance of \( D \) can be computed from its first two moments \( V(D) = \mu_2(D) - (\mu_1(D))^2 \). For some distributions the CDF can be computed from the first few moments.

Computing moments in FHE is a hard problem. In fact, even computing the first moment, i.e. the average, is hard. The main problem is the lack of a division operation in a ring. Even if a multiplicative-inverse exists, multiplying by the inverse does not act as division. For example, computing the average of 2 and 3, \( Avg(2,3) = \frac{2+3}{2} = 2.5 \), over the ring \( \mathbb{Z}_7 \). By multiplying by \( 2^{-1} \) we would have \((2 + 3) \cdot 2^{-1} = 6 \mod 7 \), where in fact we would like to have (since we are rounding things to integers) \( Avg(2,3) = 2 \). The issue of computing an average has been addressed before. In [30] it is suggested to return the pair \((\sum x_i, n)\) as the output of an average function. This solves the division problem by deferring it to a later step. This has two obvious disadvantages: (i) it requires an expensive step to perform the division either by an iteration with a Turing machine or by embedding a division polynomial and (ii) in most interesting applications we have \( \sum d_i \gg \max d_i > Avg(d_1, \ldots, d_n) \) which requires operations on numbers as large as \( \sum d_i \) which is inefficient when \( n \) is large.

**Finding \( k \) smallest values** in a general unordered array \( D = \{d_1, \ldots, d_n\} \), with \( d_i \in [L] \), for \( 1 \leq i \leq n \), is the problem of returning \( \min^k D \), where \( \min^k D \subset D \) is a subset such that \( |\min^k D| = k \) and max \( \min^k D \leq \min(D \setminus \min^k D) \). It is believed this is a hard problem to solve practically in FHE. Informally, we expect the polynomial degree of such a solution to be at least \( O(n \log L) \) as it at least needs to perform \( n - 1 \) comparisons each of \( O(\log L) \) bits. Setting \( D_1 = D \) and iteratively computing \( m_i = \min D_i \) and setting \( D_{i+1} = D_i \setminus \{m_i\} \), for \( i = 1, \ldots, n \) can be used to sort \( D \), a problem that has received a lot of attention in homomorphic encryption [11, 12, 13, 14] and no efficient solution is yet known (see more in Section 2.3).

### 1.1 Our Contribution

In this work we show, counter to common belief, that averages and higher moments can be computed efficiently in FHE and be used efficiently to solve the \( k \)-nearest neighbors problem in near real time. More specifically our contributions are as follows.

**First FHE algorithm to compute statistical functions** in a practical way by a polynomial whose degree is \( O(\deg(<)) \), the degree of a polynomial comparing encrypted numbers, and independent of \( n \). Our algorithm is also “embarrassingly parallel”, which makes it easily scalable. The output of our algorithm is an approximation of \( \frac{1}{m} \sum_{i=1}^n d_i^e \). For the applications in this paper we consider the cases where \( m = n \) and \( e \in \{1,2\} \), but it can be extended to any \( 0 \leq e \in \mathbb{R} \) and \( m > 0 \). The output of our algorithm is a single ciphertext. This has two advantages over previous techniques, whose output was the pair \( (\sum d_i^e, m) \): (i) we avoid explicitly keeping \( \sum d_i^e \) which is expensive when \( n \) is large and (ii) the output ciphertext can be passed
to a second step of an algorithm. This is in contrast to prior solutions (e.g. [30]). We expect this technique will prove efficient in more algorithms.

A novel technique for coin tossing in FHE. Our solution is non-deterministic and relies on “coin tosses” with probability that depends on an (encrypted) input. To the best of our knowledge, this is the first efficient implementation of a coin toss. Since coin tossing is a basic primitive for many random algorithms, we expect our implementation of coin tossing to have a large impact on future research in FHE.

First practical solution to compute $k$-nearest neighbors in FHE. Our techniques lead to an algorithm and a single-round protocol where the user sends one message whose size is proportional to the representation of one point $q$, and receives one message whose size is proportional to the representation of $k$ points that are nearest to $q$. The $k$-nearest neighbors algorithm has application in searching as well as in machine learning for classification and regression. A practical solution to this problem opens the way to many practical systems that employ it.

System and experiments results for search. We implemented our algorithms into a system and ran it on MIT OpenStack cloud. Our experiments, coinciding with our analysis, show that with a single core the average of 60,000 7-bit ciphertexts can be computed in about two hours. Also, finding 10 nearest neighbors in a set of 580 hotels in Boston area rounded to a 2D grid of $100 \times 100$, took about 20 minutes on a single server.

Open Source Library for Secure-Search with FHE is provided for the community [34], to reproduce our experiments, to extend our results for real-world applications, and for practitioners at industry or academy that wish to use these results for their future papers or products.

1.2 Novel Techniques

Coin tossing in FHE. The notion of coin tossing is somewhat not trivial in arithmetic circuit model, that can only realize polynomials. Since a polynomial is restricted to $+$ and $\times$ operations it is not clear how or why it is allowed to simulate a coin toss, let alone, when the probability of the toss depends on an encrypted input value. Unlike traditional RAM model algorithms, polynomials cannot “call” a sub-polynomial and cannot “choose” a sub-polynomial to embed. Instead, the process of creating such a polynomial should be such that, independently of the input, a polynomial is constructed. During the construction of the polynomial randomness is used to decide on the sub-polynomials that are embedded. We describe our algorithms as pseudo-code that get a random generator $R$ generating random $r$ parameters for coin tossing. This is of course just for clarity of presentation. Formally speaking, the random numbers are used during the construction of the polynomial to determine how it is built.

We show an equivalent of a coin toss.

Lemma 1.2 (Coin Toss) Let $n$ be a integer parameter and $D$ a distribution over $\mathbb{N}$ with a cumulative distribution function $CDF : \mathbb{N} \mapsto [0,1]$. There exists a set of polynomials $F = \{P_1, \ldots, P_n\}$, such that if $P \leftarrow F$ is drawn randomly and $x < n$ then $P(x) = 1$ with probability $CDF(x)$, and $P_i$ has degree $O(\log n)$ and size $O(\log n)$, for every $i = 0, \ldots, n$.

Using coresets to compute statistical functions. Given $n$ positive integers, $x_1, \ldots, x_n$ and $m, e > 0$ we show how to compute the mean $s^* = \frac{1}{m} \sum_{i=1}^{n} x_i^e$ in an efficient way that avoids the need of division and does not generate unnecessary large numbers. Some interesting cases for our technique are:

- take $m = n, e = 1$ to compute the average $\frac{1}{n} \sum_{i=1}^{n} x_i$.
- take $m = n, e = 2$ to compute $\frac{1}{n} \sum_{i=1}^{n} x_i^2$. 

4
We make use of our new coin tossing technique and compute the polynomial \( s = \sum a_i \), where

\[
a_i = \begin{cases} 
1 & \text{with probability } \frac{x^i}{m} \\
0 & \text{otherwise.}
\end{cases}
\]

It is easy to see that \( E(s) = s^* \). We give a bound on the probability of error \( Pr(|s - s^*| > \varepsilon s^*) < 2\exp(-\frac{\varepsilon^2 s^*}{2}) \).

Unlike previous technique, e.g. [30], our proposed solution for computing \( \frac{1}{m} \sum x_i \), enjoys the best of both worlds: keeping the ring size small \( p = \frac{1}{m} \sum x(i)^e + O(1) \), while still having a polynomial with small degree.

1.3 Definitions

\textbf{Notation.} For a given integer \( p \geq 1 \) we define \([p] = \{1, \cdots, p\}\). We denote by \([x]\) the encrypted value of \( x \), which can be a result of encrypting \( x \) by calling \( \text{encrypt}(x) \), or a result of secure computation, for example by the cloud, in which case \( x \) is not known. We also denote by \([x]_{m} \) the encryption of \( x \) given in binary representation, where each bit is encoded in a different ciphertext. As common in cryptography, the arithmetic operations in this paper are applied over a finite set of integers in \( \{0, \cdots, p - 1\} \), where the modulo operator is used to keep the outcome of each operation in this set. Such a set is formally called the \( \mathbb{Z}_p \) ring.

\textbf{Definition 1.3 (Ring)} The ring \( \mathbb{Z}_p \) is the set \( \{0, \cdots, p - 1\} \) equipped with multiplication (\( \cdot \)) and addition (\( + \)) operations modulo \( p \), i.e., \( a \cdot b = ((a \cdot b) \mod p) \) and \( a + b = ((a + b) \mod p) \) for every \( a, b \in \mathbb{Z}_p \).

We use the operator \( a \leftarrow [n] \) for a random variable \( a \) and an integer \( n \), to indicate that \( a \) is being drawn randomly from \([n]\). If \( a \) is drawn with equal probabilities we use the notation \( a \leftarrow \mathcal{U} [n] \).

2 Related Work

It has been shown \cite{23, 24, 38} that two parties (Alice and Bob) can compute any function over their secret inputs over an interactive protocol that reveals no information beyond the function’s output. The major disadvantage of an interactive protocol is the communication complexity that grows polynomially in the time to compute the function. Using an FHE scheme such as \cite{6, 22}, a single-round protocol can be devised, where Alice sends Bob her input encrypted with her public key, Bob computes the function using FHE, and returns the result to Alice. Since Alice’s secret key is needed to decipher the input and output, Bob cannot learn anything on the input or the output. On the other side, Alice does not learn anything on Bob’s input except for the output of the function.

2.1 Searching

Prior work on low degree polynomial solving Private Information Retrieval \cite{7, 10, 17, 20, 21, 33} can be used to solve the \( k \)-nearest neighbors by iteratively applying them to to search for potential neighbors in increasing order. However, these works assume there is at most one matching record. A recent work \cite{2} removed this assumption and supports any number of matches. In this case, only the first match is returned. This work also supports a filter that can be given by Alice as part of the (encrypted) input such as \( t_1 < \|q - s_i\| < t_2 \) (i.e., points whose distance from \( q \) is at least \( t_1 \) and at most \( t_2 \)). Using this filter can reduce the number of iterations. A slightly modified filter (setting a lower limit on the index of the matched record) can be used to iteratively get all matches and not only the first one. Nevertheless, this still requires at least \( k \) iterations to report the \( k \) nearest neighbors to \( q \). Another recent work \cite{11} assumes an upper bound \( s \geq 0 \) on the number of matches to an (encrypted) filter and returns all matches in an output whose size is \( O(s) \). This work draws a link between FHE and sketches and uses a sketch matrix to efficiently compact the vector of matches to a vector of dimensionality \( O(s) \). Although, it is possible to return all \( k \) records matching a filter \( \|q - s_i\| < t_2 \) in a single round, Alice is still left with the problem of finding the correct value \( t_2 \).
2.2 Classification

The problem of classification is to preprocess a set of training classified samples so that given a new sample, its address can be predicted. The problem is split into two steps. The first is the training of a model (either supervised or not). The second is to determine the classification of a new sample given a trained model. The \( k \)-nearest neighbors problem has been used for classifications in a non-secure settings. For example, in [3, 19] \( k \)NN was used to classify tissues and cancer. For some models the training step has been implemented in the secure settings. For example, decision trees [29], Naive Bayes [35] and linear discriminant classifiers [18]. Given a trained model, it was shown in [3] how to use FHE to compute medical prediction functions from patient’s encrypted data. In another work [5] secure protocols were shown to compute classification for hyperplane decision, Nave Bayes, and decision trees. However, their work uses interactive protocols (i.e. more than one round).

2.3 Sorting

If we consider the vector of distances \( D = (d_1, \ldots, d_n) \), where \( d_i \) is the distance from a query point \( q \) to a point \( s_i \), then \( k \)-nearest neighbors is closely related to sorting. Specifically, after sorting the elements of \( D \), the \( k \) smallest values correspond to the \( k \) nearest neighbors. On the other hand, an array (or positive values) can be sorted if we iteratively find the smallest number by setting \( k = 1 \) in a 1-dimensional \( KNN \) and remove it from the array. The problem of sorting an array has been greatly studied, for example in [13, 14]. In [11, 12], Cetin et al. surveyed several sorting algorithms implemented in FHE, and compared them to their arithmetic circuit that sorts \( n \) \( L \)-bit numbers, given in their binary representation. The multiplicative depth of their circuit is \( O(\log n + \log L) \), i.e. the underlying polynomial is of degree \( O(nL) \). Although their algorithm was faster than previous result, it still took almost one minute to sort 64 \( 8 \)-bit numbers.

3 Overview

**Generic Polynomials** We note that for any function \( f : [p] \rightarrow [p] \) there exists a polynomial \( P_{f,p} : \mathbb{Z}_p \rightarrow \mathbb{Z}_p \) such that \( P_{f,p}(x) = f(x) \), for any \( x \in [p] \). Moreover, the polynomial \( P_{f,p}(x) = \sum_{i=0}^{p-1} a_i x^i \) can be constructively built by interpolating over all pairs \((x, f(x))\). We can use Fermat’s little theorem to keep the degree low, \( \deg(P_{f,p}) \leq p \), since \( x^p = x \). When the ring size \( p \) is clear from the context we omit it and simply write \( P_f \). A general \( p \)-degree polynomial has size \( O(\sqrt{p}) \). See [31]. The generalization of this, a \( k \)-variate polynomial \( P_{f,p}(x_1, \ldots, x_k) \) has degree \( (p - 1)k \), and size \( O(p^{k/2}) \).

Throughout the paper we use some special polynomials: \( P_{\text{sqrt},p}(x) = \lfloor \sqrt{x} \rfloor \), and \( P_{\text{bit},p}(x) \) the \( i \)-th bit in the binary representation of \( x \), i.e.

\[
P_{\text{bit},i}(x) = \begin{cases} 
1 & \text{The } i\text{-th bit in the binary representation of } x \text{ is set} \\
0 & \text{otherwise.}
\end{cases}
\]

**Coreset** is a data summarization \( C \) of a set \( P \) of items such that \( f(C, q) \) approximates \( f(P, q) \) for a desired function \( f \) and a query \( q \) from a (usually infinite) set of queries. Coreset is a paradigm in the sense that its exact definition, structure and properties change from paper to paper. Many coresets were recently suggested to solve main problems in computational geometry (e.g. [3, 15, 32]), machine learning (e.g. [8, 27]), numerical algebra [37] and graph theory (e.g. [16]). Recently, coresets were used to solve searching problems in FHE [1, 2].

We make two extensions to coresets. The first extension is we allow the algorithm applied on \( C \) be different than \( A \). I.e. we require that \( |A'(C) - A(S)| < cA(S) \), for some \( A' \). The second extension, which is more related to FHE is we are not interested only in cases where \( |C| < |S| \), but also where \( A'(C) \) is easier to compute in FHE.
Definition 3.1 (FHE coreset) Given a vector \( S = (x_1, \ldots, x_n) \) and an algorithm \( A \), a \((\delta, \varepsilon, A', p)\) coreset is a vector \( C = (c_1, \ldots, c_m) \), where \(|A'(C) - A(S)| < \varepsilon A(S)\) and \( c_j = P_j(C) \), where \( P_j \) is a polynomial of degree \( \delta \), with probability \( p \).

**Sketch Matrix** We use a sketch matrix \( M \) that enables the client to quickly extract \( \chi \) from the small coreset (sketch) vector \( M \cdot \chi \) that is computed on the server. The coreset is based on what is known as a sketch or disjunct matrix compressing sparse vectors; see [1, 8] and references therein.

Definition 3.2 (Sketch matrix [1]) Let \( s, k, n \geq 1 \) be integers. A matrix \( M \in \{0,1\}^{k \times n} \) is called an \((s,n)\)-sketch matrix, if the following holds. There is an algorithm \( \text{DECODE} \) that, for every vector \( \chi' \in \mathbb{R}^k \), returns a binary vector \( \chi = \text{DECODE}(\chi') \in \{0,1\}^n \) if and only if \( \chi' = M \cdot \chi \). The vector \( \chi' \in \mathbb{R}^k \) is called the \( s \)-coreset of the vector \( \chi \).

Such a sketch matrix \( M \) can be efficiently constructed (offline, independently of the input) using the result of Indyk, Ngo and Rudra [28]. They suggest a probabilistic and deterministic construction that can also be decoded in time that is only poly-logarithmic in the length \( n \) of the input vector. For simplicity, we use the following less efficient but deterministic version.

Theorem 3.3 (Corollary C.3 in [28]) Let \( s, n \geq 1 \) be integers. An \((s,n)\)-sketch report matrix \( M \in \{0,1\}^{s \times n} \) (see Definition 3.2) can be computed deterministically in \( n^{O(s \log s / \log n)} \) time, and \( k^{O(1)} \) space, where \( k = O(s^2 \log n) \). The corresponding decoding algorithm computes \( \text{DECODE}(y) \) in \( k^{O(1)} \) time.

4 Problem Statement

**k Nearest Neighbors** Given a set of points \( S = \{s_1, \ldots, s_n\} \subset \mathbb{R}^d \), a query point \( q \in \mathbb{R}^d \) and a parameter \( 0 < k < n \), the search problem is to find a subset \( S^* \subset S \) such that \(|S^*| = k \) and \( \max_{\sigma \in S^*} \|\sigma - q\| \leq \min_{\sigma \in S \setminus S^*} \|\sigma - q\| \).

In the secure version, we assume two parties Alice and Bob, where Bob holds the set \( S \) of size \( n \) (for example, the set of all restaurants in the world), and Alice wants to get from Bob a short list (of size \( k \ll n \)) of restaurants nearest to her location, without having to download the more than \( O(k) \) bits from Bob, or revealing to Bob where she is. In our model, we assume Bob is semi-honest (i.e. he follows the protocol) and our goal is to prevent Bob from learning anything on the location of Alice. Furthermore, we are interested in a non-interactive protocol, meaning that Alice sends a single message of size \( O(1) \) to Bob and gets a single reply from Bob of size \( O(k) \).

5 \( k \)-nearest neighbors Algorithm

In this section we show how to solve the secure \( k \)-nearest neighbors problem in an efficient scalable way. We build upon an algorithm, which is interesting in its own and described in Section 6, to securely compute statistical functions. We also give three applications that use \( k \)-nearest neighbors.

5.1 \( k \) Nearest Neighbors (KNN) with Applications

We first start by showing 3 applications of KNN, then we show how the KNN problem can be solved by finding the \( k \) smallest values in an array. Then we provide some applications.

**Find \( k \) sites near a location** Given \( n \) sites \( S = \{s(1), \ldots, s(n)\} \subset [L]^2 \) with their associated information \( aux(1), \ldots, aux(n) \in \{0, \ldots, 2^n\} \), and an encrypted query \( [q] \in [L]^2 \) a call to \( \text{KNearestNeighbors}_{p,n,k,\varepsilon,R,s,aux}([q], [q^{(bits)}]) \) returns a sketch of the information of the \( k \)-nearest sites in \( S \) to \( q \).
Algorithm 1: KNearestNeighbors\(_{p,n,k,\varepsilon,R,d,s,\alpha,\text{aux}}([q],[q^{\text{bits}}])\)

**Parameters:** A prime \( p \), and integers \( n \gg k > 0 \), and \( d, \alpha \geq 1 \).
**Parameters:** A real number \( \varepsilon > 0 \).
**Parameters:** A family of uniform random generator functions \( R = \{R_r : \mathbb{N} \to [r]\} \).
**Parameters:** An \( n \)-tuple \( s \), where each tuple is a \( d \)-dimensional point, where \( 0 \leq s_i(i) \leq \sqrt{\frac{p}{d}} \).
**Parameters:** An \( n \)-tuple \( \text{aux} \) of auxiliary data associated with \( s \), where each tuple is an \( \alpha \)-bit number.

**Input:** A \( d \)-dimensional vector \([q] = ([q_1],\ldots,[q_d]),\) where \( 0 \leq q_\delta \leq \sqrt{p}/d \).

**Input:** The bit representation \([q^{\text{bits}}] = ([q_1^{\text{bits}}],\ldots,[q_d^{\text{bits}}])\), where \([q_\delta^{\text{bits}}]\) are the bits of \( q_\delta \).

**Output:** A sketch of size \( O((1 + \varepsilon)k) \) from which the auxiliary data of at least \( O((1 - \varepsilon)k) \) nearest neighbors can be decoded with probability \( 2^{-\varepsilon} \).

```plaintext
/* Also denote \( s_\delta = (s_\delta(1),\ldots,s_\delta(n)) \) an \( n \)-dimensional vector, where \( 0 \leq s_\delta(i) \leq \sqrt{p}/d \). */
da[i] = (s_1(i),\ldots,s_d(i))
/* Compute an \( n \)-dimensional vector of \( L_1 \)-distances from \( q \) to \( s(i) \) */
distances := [(0,\ldots,0)]
for each \( i \in 1,\ldots,n \) do
  /* Define: \( ||a|| - b||_{L_1} = \sum_\delta(b_\delta - \|a_\delta\|)\)\(\textit{isSmaller}_p([b^{\text{bits}}],[a^{\text{bits}}])\) */
distances(i) := ||[q] - s(i)||_{L_1}
distances^{\text{bits}}(i) := Convert \( [\text{distances}(i)] \) to bit representations using the polynomials \( \mathbb{P}_{\text{bit},p} \)
/* KSmallestValues computes an indicator vector for the \( O(k) \) least distances. See Algorithm 2. */
\[ [\chi] := \text{KSmallestValues}_{p,n,k,R}([\text{distances}^{\text{bits}}]) \]
for each \( i \in 1,\ldots,n \) do
  for each \( \delta \in 1,\ldots,\alpha \) do
  \[ \text{KNN}_\delta^{\text{bits}}(i) := \text{aux}_\delta^{\text{bits}}(i)\chi(i) \]
/* Use a sketch to return the sparse result vector. See Section 3 and 11 for details */
\[ \text{Result}^{\text{bits}} := [((0,\ldots,0),\ldots,(0,\ldots,0))] \] \(\text{a} (1 + \varepsilon)k\)-dimensional vector of \( d \)-tuples
for each \( \delta \in 1,\ldots,\alpha \) do
  \[ \text{Result}_\delta^{\text{bits}} := M_{(1+\varepsilon)\times n} \cdot \text{KNN}_\delta \]
Return \([\text{Result}]\)
```

8
Classification System Given $n$ samples $S = \{s(1), \ldots, s(n)\} \subset [L]^d$ with their associated classes $aux(1), \ldots, aux(n) \in \{0, 1\}$, and an encrypted query $[q] \in [L]^d$ a call to KNearestNeighbors\(_{p,n,k,\varepsilon,R,s,aux}([q], [qubits])\) returns a sketch of the classification of the $k$-nearest samples in $S$ to $q$. The classification of $q$ is then taken to be the majority of classifications.

Recommender system Given $n$ profiles $S = \{s(1), \ldots, s(n)\} \subset [L]^d$ with recommendations associated to them $aux(1), \ldots, aux(n) \in \{0, \ldots, 2^n\}$, and an encrypted query profile $[q] \in [L]^d$, a call to KNearestNeighbors\(_{p,n,k,\varepsilon,R,s,aux}([q], [qubits])\) returns a sketch of the recommendations of the $k$-nearest profiles in $S$ to $q$. The set of recommendations to $q$ is then taken to be the set of recommendations to profiles similar to $q$.

Algorithm Overview Algorithm\(_1\) has (as parameters) integers $n \gg k > 0$, a prime, $p$ and an $n$-tuple, $s$, where each tuple is a $d$-tuple $(s_1(i), \ldots, s_d(i))$ denoting the position of the $i$-th site in $d$-space, where $0 \leq s_x(i), s_y(i) \leq \sqrt{p}/d$ are integers. The algorithm gets as input an encrypted query $q = (q_1, \ldots, q_d)$, where $0 \leq q_i \leq \sqrt{p}/d$ denotes the query location given in binary representation as well as a single ciphertext, and it outputs a sketch of size $O(k)$ from which the user can infer $\{s(i^*) \mid q - s(i^*)|_{L_1} \in \min_k \{||q - s(i)||\}\}$, that is, the set of sites that are nearest to $q$, where $||p_1 - p_2||_{L_1}$ is the distance in $L_1$ norm.

In Line 3 Algorithm\(_1\) computes the distance $||d_1 - d_2||_{L_1} = ||d_1 - d_2||_{L_1}$, for each $1 \leq i \leq n$. The $L_1$ norm is computed using Algorithm\(_5\) $||a - b||_{L_1} = \sum \delta (a - b)_i = \sum \delta (a - b + 2(b - a)) isSmaller(b^{\text{bits}}, a^{\text{bits}})$.

In Line 5 the distance $||d_1 - d_2||_{L_1}$ is converted to bit representation. This is done by computing $[\log p]$ polynomials $P_{bit, p}(distances(i))$, $P_{bit, p}(distances(i))$, where $P_{bit, p}(x) = 1$ if the $i$-th bit is set in the binary representation of $x$. See Section 3 In Line 5 Algorithm\(_1\) calls Algorithm\(_2\) to get an encrypted indicator vector $[\chi]$, where $\chi(i) = 1$ if $s(i) \in \min_k s$ and $\chi(i) = 0$ otherwise, for some $k = O(k)$. In Line 8 the algorithm creates a set of sparse vectors by multiplying the elements of $aux$ by $\chi$. This guarantees that only the auxiliary data of the $k$ nearest neighbors appear in $KNN$. To be generic, we have $aux(i)$, the auxiliary associated with the $i$-th site, be $\alpha$ bits, where $\alpha$ is a parameter. We keep the result of multiplying the bits of $aux$ by $\chi$ in $\alpha n$-dimensional binary vectors. By the way Algorithm KSmaldestValues constructs $\chi$, we get that it (and also $KNN$) is $(1 + \varepsilon)k$-sparse. In Line 11 we multiply $KNN$ by a precomputed constant sketch matrix $M$ to get a $(1 + \varepsilon)k$ dimensional sketch of $KNN$.

Theorem 5.1 Given $n$ points $s(1), \ldots, s(n) \in [\sqrt{p}/d]^{\alpha}$, an $n$-tuple of auxiliary data $aux$, such that $aux(i)$ is an $\alpha$-tuple of bits associated with $s(i)$ and a query point $q \in [\sqrt{p}/d]^{\alpha}$. Denote by $D = \{d(1), \ldots, d(n)\}$ the set of $L_1$ distances, $d(i) = ||s(i) - q||_{L_1}$. For any $0 \leq k \leq n$, Algorithm\(_1\) outputs a compact representation of $\{aux(i) \mid d(i) \in \min_k D\}$ of size $O(k)$ of the auxiliary data associated with the $O(k)$ nearest neighbors of $q$. The polynomials that compute the sketches have degree $O(\log n)$ and size $O(\sqrt{p}\log p + nd\log p + \alpha n)$.

The proof is given in Section 7.5

5.2 Finding $k$ smallest values in a set

As mentioned in the introduction, the problem of finding the $k$ smallest values in a set $D$ is a hard problem in FHE. We relax this problem by assuming the set $D$ has a known distribution and specifically we show here a solution for sets with Gaussian distribution. Another relaxation we make, given $D$ and $k$, is to compute $\min_k D$ where $k \approx k$ (see a more formal relation below).

Algorithm\(_2\) Overview Algorithm\(_2\) has as parameters two integers $n \gg k > 0$ and a prime $p$, where $n$ is the size of the input and $k$ is the size of the output (i.e. number of smallest values to report). The algorithm gets as input an encrypted $n$-dimensional vector $d \in [p]^n$. The algorithm assumes the values of $d$ have Gaussian distribution, $(d(1), \ldots, d(n)) \sim N(\mu, \sigma^2)$ (where $\mu$ and $\sigma$ are unknown). For many applications this is a valid assumption. For example, our experiments (see Section 8.1.3) show that the
Algorithm 2: KSamplesValues_{p,n,k,R([d^{\text{bits}}])}

**Parameters:** Two integers, $n \gg k > 0$, and a prime $p$.

**Parameters:** A random generator functions $R = \{ R : \mathbb{N} \rightarrow [n] \}$.

**Input:** Encrypted $n$-dimensional vector $[d^{\text{bits}}]$, where $d \in [\sqrt{p}]^n$.

**Output:** An $n$-dimensional indicator vector $[\chi]$, where $\chi(i) = 1$ if $d(i) \in \min^k \{ d(i) \}$ and $\chi(i) = 0$ otherwise, where $k \approx n$ with high probability (see details below).

1. /* Compute an estimation of $\frac{1}{n} \sum d_i$. See Section 6 for details. */
   \[ [X] := \text{ProbabilisticAverage}_{p,n,1,\sqrt{p},R([d^{\text{bits}}])} \]
2. /* Compute an estimation of $\frac{1}{n} \sum d_i^2$ */
   \[ [X^2] := \text{ProbabilisticAverage}_{p,n,2,\sqrt{p},R([d^{\text{bits}}])} \]
3. /* Compute estimation of the deviation */
   \[ [\sigma^*] := [\sqrt{[X^2]} - [X]^2] \]
4. /* Compute a threshold for the report */
   \[ [T^*] := [X] + \Phi(\frac{k}{n})[\sigma^*] \]
5. /* Compute an indicator vector for the report */
   \[ [\chi] := (0, ..., 0) \]
6. /* Compute an estimation of $\frac{1}{n} \sum d_i$ by calling ProbabilisticAverage, with $e = 1, m = n, l = \sqrt{p}$ (see Section 6 for more details). */
   \[ [T^*] = \text{ProbabilisticAverage}_{p,n,1,\sqrt{p},R([d^{\text{bits}}])} \]
7. /* Compute the average of squares $\sum d_i^2$ by calling ProbabilisticAverage, with $e = 2, m = n, l = \sqrt{p}$ (see Section 6 for more details). */
   \[ [X^2] = \text{ProbabilisticAverage}_{p,n,2,\sqrt{p},R([d^{\text{bits}}])} \]
8. /* We approximate the standard deviation by computing $[\sigma] = [\sqrt{[X^2]} - [X]^2]$, where $\sqrt{\cdot}$ is computed by applying $\text{ProbabilisticAverage}$. */
   \[ \chi(i) := (0, ..., 0) \] // an $n$ dimensional vector
9. /* We approximate the square of distances indeed has Gaussian distribution. The algorithm returns an (encrypted) indicator vector $\chi \in \{0, 1\}^n$, such that $\chi(i) = 1$ if $d(i) \in \min^k \{ d(1), ... , d(n) \}$, where $Pr(|\kappa - k| < \varepsilon n (\frac{\mu}{\sigma} + \Phi^{-1}(\frac{k}{n})) \cup 2 exp \left( -\varepsilon O \left( \frac{\mu^2}{\sigma^2} + \mu \right) \right)$, where $\Phi(\tau) = \frac{|\{d | d < \tau\}|}{n}$ is the cumulative distribution function. We distinct parameters whose values are known to the algorithm from input that are encrypted. Given the values of $\mu$ and $\sigma$ it easy to compute $T$. Unfortunately, as discussed in Section 7 these values are hard to compute in arithmetic circuit model. */

Theorem 5.2 Given an array $D = (d(1), ..., d(n))$, of $n$ elements, where $0 < d(i) < \sqrt{p}$, with Gaussian distribution, $D \sim \mathcal{N}(\mu, \sigma^2)$ and a parameter $k \ll n$, then KSamplesValues evaluates an indicator vector $\chi = (\chi(1), ..., \chi(n))$, such that $\chi(i) = 1$ if $d(i) < \max^k D$, where

\[
Pr \left( |\kappa - k| < \varepsilon n \sqrt{2\pi} \left( \frac{\mu}{\sigma} + \Phi^{-1}(\frac{k}{n}) \right) \right) < 2 exp \left( -\varepsilon O \left( \frac{\mu^2}{\sigma^2} + \mu \right) \right),
\]

where $\Phi(\tau) = \frac{|\{d \mid d < \tau\}|}{n}$ is the cumulative distribution function of standard Gaussian distribution. Also, $\chi$ can be computed by a polynomial of degree $O(\log p)$ and size $O(n \log p)$.

The proof is given in Section 7.4.

In the next section we describe how $\overline{X}$ and $\overline{X^2}$ are computed efficiently in arithmetic circuit model.
6 Algorithms for Computing Moments and Statistical Functions

In this section we show how statistical functions such as $\frac{1}{n} \sum x_i$ and $\frac{1}{n} \sum x_i^2$ and other moments can be computed efficiently under FHE. We believe computing these functions is of independent interest and it has a broad set of applications expanding beyond k-nearest neighbors.

Algorithm 3 Overview

In Line 1 Algorithm 3 computes a resampling constant to ensure $\frac{1}{m} \sum x_i < 1$ for any $i$. See details below. In Line 4 the algorithm uses $cn$ values $r_1, \ldots, r_n$ drawn uniformly from $[cm]$, which are used to set

$$a_i = \begin{cases} 1 & \text{if } (r_i)^{1/e} < x[i/c] \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm then returns $\sum a_i$ as an estimation to $\frac{1}{m} \sum x_i^e$.

Theorem 6.1 Let $(x(1), \ldots, x(n)) \in [L]_n$ be an $n$ dimensional vector. Also let $0 < e \in \mathbb{R}$ and $m$ be an integer. Denote $\mu = \frac{1}{m} \sum x_i^e$, then Algorithm 3 describes a polynomial whose value is $X^*$, that has degree $O(\log L)$ and size $O([\frac{L}{m}]n \log L)$ such that

$$Pr(|X^* - \mu| \leq \epsilon \mu) < 2e^{-\min}(\epsilon \mu^2) \cdot 3).$$

The proof is given in Section 7.3.

Connection to coresets Our algorithm for computing statistical functions is in fact an FHE-coreset (see Definition 3.1) with $A(S) = \frac{1}{n} \sum x_i^e$, for $S = (x_1, \ldots, x_n)$. $A'(C) = \sum c_i$, for $C = (c_1, \ldots, c_n)$, and $c_j = \text{CoinToss}_{L_j}(x_j)$, which is a polynomial of degree $L$, and $|A'(C) - A(S)| < \epsilon A(S)$, with high probability.

6.1 Coin Toss

CoinToss Overview The formal way to look at CoinToss$_{n,r}$ is as an infinite set of algorithms, with one algorithm for every pair of $(n, r)$ for any $n, r \in \mathbb{Z}$. Setting $n$ and $r$, the algorithm is uniquely determined and it gets an encrypted input $[x_i^{\text{bits}}]$, where $0 < x < n$, given by its binary representation, i.e. as a vector of bits, where $[x(i)]$ is the $i$-th bit of $x$. Since $n$ and $r$ are “embedded” into the algorithm their values are known (not encrypted) and therefore can be used in a comparison operation. In Line 2 the algorithm tests whether $r > n \geq x$, and returns $[0]$ if it is. Although a polynomial does not have a comparison operator,
Lemma 6.2 The algorithm \( \text{CoinToss}_{n, r}([x^{\text{bits}}]) \), where \( 0 \leq x < m \) and \( r \overset{D}{\leftarrow} [m] \) is drawn from \([m]\) with distribution \( D \), returns a bit \( b \), such that \( \Pr(b = 1) = \text{CDF}_{D}(x) \), where \( \text{CDF}_{D}(x) = \Pr(r < x) \) is the cumulative distribution function of \( D \), and the polynomial has degree \( O(\log m) \) and size \( O(\log m) \).

The proof for this lemma is given in Section 7.2.

6.2 Comparing Encrypted Values

In this subsection we show, for completeness, how to compare two encrypted values, given by their binary representation.

Algorithm 5: \( \text{isSmaller}_{n}([x^{\text{bits}}], [y^{\text{bits}}]) \)

**Parameters:** An integer \( n > 0 \).

**Input:** Two encrypted numbers \([x], [y]\), such that \( 0 \leq x, y < n \) are given in their binary form and \( x(i) \) and \( y(i) \) denote their \( i \)-th bits.

**Output:** Return \([1]\) iff \( y < x \), and \([0]\) otherwise.

```plaintext
/* Denote \( x(i) \) and \( y(i) \) the \( i \)-th bit of \( x \) and \( y \), respectively */
samePrefix := ([1], [0], ..., [0])
for each \( i = 1, \ldots, \lceil \log n \rceil \) do
    sameBit(i) := 1 - ([x(i)] - [y(i)])^2
    samePrefix(i + 1) := \( \prod_{j=1}^{i} \text{sameBit}(j) \)
return \( \sum_{i=1}^{\lceil \log n \rceil} (1 - y(i))x(i)\text{samePrefix}(i) \)
```

**isSmaller Overview** The isSmaller algorithm gets a parameter \( n \), and two encrypted input values \([x]\) and \([y]\), given by their binary representation, i.e. as a vector of bits, where \([x(i)]\) and \([y(i)]\) are the \( i \)-th bit of \( x \) and \( y \), respectively. In line 3 the algorithm computes a binary \( n \)-dimensional vector \( \text{sameBit} \) such that \( \text{sameBit}(i) = 1 \) iff \( x(i) = y(i) \). In line 4 the algorithm computes a binary \( n \)-dimensional vector \( \text{samePrefix} \) such that \( \text{samePrefix}(i) = 1 \) iff at least \( i - 1 \) most significant bits of \( x \) and \( y \) are equal by multiplying \( \text{sameBit}(j) \) for \( j = 1, \ldots, i - 1 \). The first element \( \text{samePrefix}(1) \) is set to \([1]\). At line 5 the algorithm returns \( \sum_{i=1}^{\lceil \log n \rceil} (1 - y(i))x(i)\text{samePrefix}(i) \).
Lemma 6.3 The polynomial realized by Algorithm isSmaller$_n([x],[y])$ equals 1 if $x < y$ and 0 otherwise, where $x$ and $y$ are given as $[\log n]$-bit binary number, and it is of degree $O(\log n)$, and size $O(\log n)$.

The proof is given in Section 7.1.

7 Analysis

In this section we prove the correctness and efficiency of our algorithms. Unlike the algorithms that were presented top-down, reducing one problem to another simpler problem, we give the proofs bottom up as analyzing the efficiency of one algorithm builds upon the efficiency of the simpler algorithm.

7.1 Analysis of isSmaller

Lemma 6.3 The polynomial realized by Algorithm isSmaller$_n([x],[y])$ equals 1 if $x < y$ and 0 otherwise, where $x$ and $y$ are given as $[\log n]$-bit binary number, and it is of degree $O(\log n)$, and size $O(\log n)$.

Proof. Notice that $1 - (x(i) - y(i))^2 = 1$ if $x(i) = y(i)$ and 0 otherwise. Therefore, samePrefix$(i) = 1$ if $x(j) = y(j)$, for all $j < i$, effectively meaning $x$ and $y$ have a prefix of at least $i$ bits. Similarly, since $x(i), y(i) \in \{0, 1\}$, we have that $(1 - y(i))x(i) = 1$ only when $0 = y(i) < x(i) = 1$.

Let $y < x$ and let $i^*$ be the index of the first bit that is not equal in $x$ and $y$, i.e. $0 = y(i^*) < x(i^*) = 1$. For all $i < i^*$ we have $(1 - y(i))x(i) = 0$, since either $x(i) = y(i) = 0$ or $x(i) = y(i) = 1$. For all $i > i^*$ we have samePrefix$(i) = 0$. Finally, we have $(1 - y(i^*))x(i^*)$ samePrefix$(i^*) = 1$. Therefore isSmaller$_n(x, y)$ evaluates to 1.

On the other way, if isSmaller$_n(x, y) \neq 0$, then for some $i^*$ we have $(1 - y(i^*))x(i^*) \neq 0$. Since $x(i), y(i) \in \{0, 1\}$ this means $x(i^*) = 1$ and $y(i^*) = 0$ which implies that samePrefix$(i) = 0$, for $i > i^*$. Also, we have samePrefix$(i^*) \neq 0$, which means that $x(i) = y(i)$ for $i < i^*$. Combining the two, we get that $y < x$.

The degree of the samePrefix$_n(x, y)$ polynomial is dominated by the degree of samePrefix, which is $O(\log n)$, Similarly, the size is $O(\log n)$.

7.2 Analysis of Coin Toss

We argue that an equivalent of a coin toss that depends on an encrypted input can be achieved.

Lemma 6.2 The algorithm CoinToss$_{m,r}([x(bits)])$, where $0 \leq x < m$ and $r \overset{D}{\rightarrow} [m]$ is drawn from $[m]$ with distribution $D$, returns a bit $b$, such that $Pr(b = 1) = CDF_D(x)$, where $CDF_D(x) = Pr(r < x)$ is the cumulative distribution function of $D$, and the polynomial has degree $O(\log m)$ and size $O(\log m)$.

Proof. Let $\mathbb{F} = \{\mathbb{F}_0, \ldots, \mathbb{F}_{m-1}\}$, where $\mathbb{F}_r(x) = \text{isSmaller}_m(r, x)$, where $\text{isSmaller}_m(r, x)$ is 1 if $r < x$ and 0 otherwise. Since $r \overset{D}{\rightarrow} [m]$ is drawn from $[m]$ with distribution $D$, independent to the choice of $x$, it follows that $Pr(\text{CoinToss}_{m,r}([x])) = Pr(r < x) = \sum_{i < x} Pr(r = i) = CDF_D(x)$. Following Lemma 6.3 we have $\deg(\text{CoinToss}_{m,r}) = O(\log m)$ and size(\text{CoinToss}_{m,r}) = $O(\log m)$.

The proof of the following lemma follows immediately.

Lemma 1.2 (Coin Toss) Let $n$ be a integer parameter and $D$ a distribution over $\mathbb{N}$ with a cumulative distribution function $CDF : \mathbb{N} \to [0,1]$. There exists a set of polynomials $\mathbb{F} = \{\mathbb{F}_1, \ldots, \mathbb{F}_n\}$, such that if $\mathbb{P} \leftarrow \mathbb{F}$ is drawn randomly and $x < n$ then $Pr(\mathbb{P}(x) = 1) = CDF(x)$, and $\mathbb{F}_i$ has degree $O(\log n)$ and size $O(\log n)$, for every $i = 0, \ldots, n$. 

13
7.3 Analysis of ProbabilisticAverage

To prove the correctness of Algorithm 3, we first prove this lemma.

**Lemma 7.1** Let $x \in [L]^n$ be an $n$ dimensional vector. Also $m > \max x(i)$ be a parameter. Denote $\mu = \frac{1}{m} \sum_i x(i)$ and set independent Bernoulli random variables:

$$a_i = \begin{cases} 1 & \text{with probability } \frac{x(i)}{m} \\ 0 & \text{otherwise.} \end{cases}$$

Then,

$$Pr \left( \sum a_i > (1 + \varepsilon)\mu \right) < \exp\left(-\frac{\mu\varepsilon^2}{3}\right),$$

$$Pr \left( \sum a_i < (1 - \varepsilon)\mu \right) < \exp\left(-\frac{\mu\varepsilon^2}{2}\right)$$

and

$$Pr \left( \left| \sum a_i - \mu \right| > \varepsilon\mu \right) < 2\exp\left(-\frac{\mu\varepsilon^2}{3}\right).$$

**Proof.** Let $x \in [L]^n$ be an $n$ dimensional vector. Since $a_i$ are independent Bernoulli random variables, we have that $E(\sum a_i) = \frac{1}{m} \sum x(i) = \mu$ and by Chernoff we have: $Pr \left( \sum a_i > (1 + \varepsilon)\mu \right) < \exp\left(-\frac{\mu\varepsilon^2}{3}\right)$ and $Pr \left( \sum a_i < (1 - \varepsilon)\mu \right) < \exp\left(-\frac{\mu\varepsilon^2}{2}\right)$, from which it immediately follows that $Pr \left( \left| \sum a_i - \mu \right| > \varepsilon\mu \right) < 2\exp\left(-\frac{\mu\varepsilon^2}{3}\right)$.

**Theorem 6.1** Let $(x(1), \ldots, x(n)) \in [L]^n$ be an $n$ dimensional vector. Also let $0 < e \in \mathbb{R}$ and $m$ be an integer. Denote $\mu = \frac{1}{m} \sum x_i$, then Algorithm 3 describes a polynomial whose value is $X^*$, that has degree $O(\log L)$ and size $O(\frac{L}{m} \log L)$ such that

$$Pr \left( \left| X^* - \mu \right| > \varepsilon\mu \right) < 2\exp\left(-\frac{\mu\varepsilon^2}{3}\right).$$

**Proof.** Let $x \in [L]^n$, $m$ and $e$ be as defined in the theorem. Assume first that $\max x(i)e < m$. If this assumption does not hold, set $c = \lfloor \frac{L}{m}\rfloor$, define $n' = cn$, $m' = cm$, $x' \in [L]^{n'}$ such that $x'_i = x(\lfloor i/c \rfloor)$ and continue with $x'$, $m'$ and $n'$.

Denote $\mu = \frac{1}{m} \sum x'_i$. Set $X^* = \sum a_i$, where $a_i$ is a Bernoulli variable with $Pr(a_i = 1) = \frac{x(i)e}{m}$. By Lemma 7.1 we have that $Pr \left( \left| X^* - \mu \right| > \varepsilon\mu \right) < 2\exp\left(-\frac{\mu\varepsilon^2}{3}\right)$, as required.

We now bound the size and depth of a polynomial that evaluates one Bernoulli random variable with probability $Pr(a_i = 1) = \frac{x(i)e}{m}$. The naive approach CoinTossL,r,([x]e), with $r_i \mathbin{\$} [m]$ would require computing $x^e$ in binary representation, a thing that increases the degree and size of the polynomial. To make Algorithm 3 more efficient, we notice that $Pr(x^e < r) = Pr(x < \sqrt[r]{n})$ and consider the distribution $D_{e,m}$ with $Pr(r = r_0) = \frac{|\{u \leq m \mid \frac{[\sqrt[r]{u}]e}{m} = r_0\}|}{m}$, that draws $r$ (non-uniformly) from $\sqrt[r]{m}$ by setting $r = \sqrt[r]{u}$, where $u \mathbin{\$} [m]$ is drawn uniformly from $\{0, \ldots, m - 1\}$. Thus setting $a_i = \text{CoinToss}_{L,r,}\([x(i)]\)$, with $r_i = \sqrt[r]{u_i}$, where $u_i \mathbin{\$} [m]$ is drawn uniformly. We get from Lemma 1.2 that $Pr(\text{CoinToss}_{L,r,}\([x[i])\) = 1) = \frac{x(i)e}{m}$. Since $X^*$ is constructed by adding $\lceil \frac{L}{m}\rceil$ outputs of CoinToss_{L,r}, the degree of ProbabilisticAverage is $O(\log L)$ and its size is $O(\frac{L}{m} \log L)$.

7.4 Analysis of KSmallestValues

**Theorem 5.2** Given an array $D = (d(1), \ldots, d(n))$, of $n$ elements, where $0 < d(i) < \sqrt[n]{p}$, with Gaussian distribution, $D \sim N(\mu, \sigma^2)$ and a parameter $k \ll n$, then KSmallestValues evaluates an indicator vector
\( \chi = (\chi(1), \ldots, \chi(n)) \), such that \( \chi(i) = 1 \) if \( d(i) < \text{max}^\kappa D \), where

\[
Pr \left( |\kappa - k| < \frac{\varepsilon n}{\sqrt{2\pi}} \left( \frac{\mu}{\sigma} + \Phi^{-1} \left( \frac{k}{n} \right) \right) \right) < 2\exp \left( -\varepsilon O \left( \sigma^2 + \frac{\mu^4}{\sigma^2 + \mu^2} + \mu \right) \right),
\]

where \( \Phi(\tau) = \left\{ \frac{i - n\tau}{n} \right\} \) is the cumulative distribution function of standard Gaussian distribution. Also, \( \chi \) can be computed by a polynomial of degree \( O(\log p) \) and size \( O(n \log p) \).

**Proof.**

Let \( \mu = \frac{1}{n} \sum d_i \) and \( \mu_2 = \frac{1}{n} \sum d_i^2 \) be the first two moments and \( \sigma^2 = \mu_2 - \mu^2 \) be the variance of \( D \). Also, let \( T = Q\left( \frac{\varepsilon}{n} \right) \), where \( Q \) is the quantile function, i.e., \( \left\{ \frac{d_i}{d_i < T} \right\} = k \). For the Gaussian distribution we have \( Q\left( \frac{\varepsilon}{n} \right) = \mu + \Phi^{-1}\left( \frac{k}{n} \right) \sigma \), where \( \Phi \) is the CDF function of the standard Gaussian distribution, and \( \Phi^{-1} \) is its inverse. Algorithm \([2]\) computes \( \overline{X} \) and \( \overline{X}^2 \), using Algorithm \([3]\) as estimations to \( \mu \) and \( \mu_2 \), respectively. Denote \( \sigma^* = \sqrt{X^2 - \overline{X}^2} \).

Set \( \varepsilon' = 2\varepsilon + (4\varepsilon^2 - 6\varepsilon)\frac{\mu^2}{\sigma^2} \), then \( Pr(\overline{X}^2 > (1 + \varepsilon')\mu_2) < \exp\left(-\frac{\mu_2\varepsilon'^2}{3}\right) \). Since, \( (1 + \varepsilon')\mu_2 - (1 - \varepsilon)^2\mu_2 = (1 - 2\varepsilon)\mu_2 - (1 + 2\varepsilon)\mu_2 \), we have:

\[
Pr \left( (\sigma^*)^2 > (1 + 2\varepsilon)\sigma^2 \right) < \exp\left(-\frac{\mu_2\varepsilon'^2}{3} + 6\mu_2\varepsilon \right) = \exp\left(-\varepsilon O \left( \sigma^2 + \frac{\mu^4}{\mu_2} + \mu \right) \right).
\]

Similarly, set \( \varepsilon'' = \varepsilon - (\varepsilon^2 + 3\varepsilon)\frac{\mu^2}{\sigma^2} \), then \( Pr(\overline{X}^2 < (1 - \varepsilon'')\mu_2) < \exp\left(-\frac{\mu_2\varepsilon''^2}{6}\right) \). Since, \( (1 - \varepsilon'')\mu_2 - (1 - \varepsilon)^2\mu_2 = (1 - \varepsilon)\mu_2 - (1 + \varepsilon)\mu_2 \), we have:

\[
Pr \left( (\sigma^*)^2 < (1 - \varepsilon)\sigma^2 \right) < \exp\left(-\frac{2\mu_2\varepsilon'^2}{6} + 3\mu_2\varepsilon \right) = \exp\left(-\varepsilon O \left( \sigma^2 + \frac{\mu^4}{\mu_2} + \mu \right) \right).
\]

Since \( \sqrt{1 + 2\varepsilon} < 1 + \varepsilon \) and \( \sqrt{1 - \varepsilon} < 1 - \varepsilon \) we get that \( Pr(|\sigma^* - \sigma| < (1 + \varepsilon)\sigma) < \exp\left(-\frac{\mu_2\varepsilon'^2 + 6\mu_2\varepsilon}{3}\right) + \exp\left(-\frac{2\mu_2\varepsilon''^2 + 3\mu_2\varepsilon}{6}\right) \), with \( \varepsilon', \varepsilon'' \) as defined above. We therefore have \( \frac{k - \kappa}{n} = CDF(T^*) - CDF(T) \). For Gaussian distribution we have \( CDF(T^*) - CDF(T) = \Phi(T^* - \mu) - \Phi(T - \mu) \). The inequality being with probability at least \( 1 - \exp\left(-\varepsilon O \left( \sigma^2 + \frac{\mu^4}{\sigma^2 + \mu} + \mu \right) \right) \). Since \( \Phi'(x) < \frac{1}{\sqrt{2\pi}} \) we have \( \Phi(a + b) < \phi(a) + \frac{1}{\sqrt{2\pi}} \) and therefore

\[
\Phi \left( \frac{(1 + \varepsilon)T - \mu}{\sigma} \right) - \Phi \left( \frac{T - \mu}{\sigma} \right) < \frac{\varepsilon}{\sqrt{2\pi}} \left( \frac{\mu}{\sigma} + \Phi^{-1} \left( \frac{k}{n} \right) \right).
\]

Similarly, we have that \( \frac{k - \kappa}{n} = CDF(T^*) - CDF(T^*) \), and for Gaussian distribution we get that \( \Phi(T^* - \mu) - \Phi(T - \mu) > \Phi(T^* - \mu) - \Phi(T - T^* - \mu) \). The inequality being with probability at least \( 1 - \exp\left(-\varepsilon O \left( \sigma^2 + \frac{\mu^4}{\sigma^2 + \mu} + \mu \right) \right) \). Since \( \Phi'(x) < \frac{1}{\sqrt{2\pi}} \) we have

\[
\Phi \left( \frac{T - \mu}{\sigma} \right) - \Phi \left( \frac{(1 - \varepsilon)T - \mu}{\sigma} \right) < \frac{\varepsilon}{\sqrt{2\pi}} \left( \frac{\mu}{\sigma} + \Phi^{-1} \left( \frac{k}{n} \right) \right).
\]

Putting it all together, we get that if we set \( T^* = \mu^* + \sigma^*\Phi^{-1}\left( \frac{k}{n} \right) \), then for \( \kappa = |\{ d(i) | d(i) < T^* \}| = \Phi(T^* - \mu) \) we have:

\[
Pr \left( |\kappa - k| < \frac{\varepsilon n}{\sqrt{2\pi}} \left( \frac{\mu}{\sigma} + \Phi^{-1} \left( \frac{k}{n} \right) \right) \right) < 2\exp \left( -\varepsilon O \left( \sigma^2 + \frac{\mu^4}{\sigma^2 + \mu^2} + \mu \right) \right).
\]

\qed
7.4.1 Extension to other distributions

Algorithm 2 and Theorem 5.2 assume the distribution of $X$ is Gaussian, however, this can be extended to a large set of distributions. To see what is needed from a distribution let us review where the assumption of the distribution of $X$ was used.

CDF with bounded derivative. To bound the number of elements in the output of Algorithm 2, we need some bound on the derivative of the quantile function, $|Q'(x)| < c$, for some $c \in \mathbb{R}$. More specifically, we are interested in such a bound in the segment $\frac{k}{n} - \delta < x < \frac{k}{n} + \delta$, where $\delta$ depends on $\varepsilon$ and the parameters of the distribution. Informally, this bound means that the error in computing the first two moments of $X$ does not change by much the number of elements in $\{d_i \mid d_i < T\}$. While such a bound exists for all distributions, the algorithm is impractical for large values of $c$. For example, for a Bernoulli distribution with $Pr(x = 1) = Pr(x = 0) = 1/2$, we have $CDF(0) = 1/2$ and $CDF(1) = 1$, which means the algorithm can either return about half of $X$ or all of $X$. Algorithm 1 assumes $k < n$. For Gaussian distribution we have $\Phi'(x) < \frac{1}{\sqrt{2\pi}}$, and for $\frac{k}{n} \leq \frac{1}{2}$ we even have $\Phi'(\frac{k}{n} + \delta) \ll \frac{1}{\sqrt{2\pi}}$ for sufficiently small values of $\delta$.

Distribution parameters easily computed from moments. The second time the algorithm assumes the distribution is Gaussian $N(\mu, \sigma^2)$ is when computing a threshold $T$ using the quantile function $T = Q(k/n)$. For general Gaussian distribution we have $Q(x) = \mu + \Phi^{-1}(x)\sigma$ where $\Phi^{-1}(x)$ is the inverse of the standard CDF for Gaussian distribution. To extend to another distribution one needs to show how to compute the quantile function for that distribution.

7.5 Analysis of KNearestNeighbors

Theorem 5.1 Given $n$ points $s(1), \ldots, s(n) \in [\sqrt{n}]^d$, an $n$-tuple of auxiliary data $aux$, such that $aux(i)$ is an $\alpha$-tuple of bits associated with $s(i)$ and a query point $q \in [\sqrt{n}]^d$. Denote by $D = \{d(1), \ldots, d(n)\}$ the set of $L_1$ distances, $d(i) = \|s(i) - q\|_{L_1}$. For any $0 \leq k \leq n$, Algorithm 1 outputs a compact representation of $\{aux(i) \mid d(i) \in \text{min}^{O(k)} D\}$ of size $O(k)$ of the auxiliary data associated with the $O(k)$ nearest neighbors of $q$. The polynomials that compute the sketches have degree $O(\log p)$ and size $O(\sqrt{p}\log p + nd\log p + \alpha n)$.

Proof. The correctness of Algorithm 1 follows by combining Theorem 5.2 with the compacting method described in [1]. More specifically, after computing a set of distances $D = \{d(1), \ldots, d(n)\}$, Theorem 5.2 states that an indicator vector $\chi$ of the smallest $k$ distances can be computed, where $k \approx n$ with high probability (see Theorem 5.2 for exact details). Using the result of [1], a compact vector $\chi' \in \{p\}^{O(n)}$ can be created, such that $\chi$ can be reconstructed from $\chi'$, since it is $k$-sparse.

Since Theorem 5.2 states that $Pr(|\kappa - k| < \varepsilon c_1 n) < 2exp(-\varepsilon c_2)$, where $c_1$ and $c_2$ are constants that depend on the distribution parameters of $D$, we get that $\chi$ can be reconstructed from $\chi'$ with that probability.

Since computing an $L_1$-distance function involves $d$ calls to isSmaller and $d$ multiplications, computing all the distances can be done by a polynomial of degree $O(\log p)$ and size $O(nd\log p)$. As argued in [1], since the sketch matrix, $M$, is a global constant, the multiplication $M\chi$ can be done without ciphertext multiplications at all.

Putting it all together, we get that Algorithm 1 can be realized by a polynomial of degree $O(\log p)$ size $O(\sqrt{p}\log p + nd\log p + \alpha n)$. \hfill \Box

Boosting probability To boost the success probability, KNearestNeighbors can be computed $b$ times independently, thus boosting the success probability. KNearestNeighbors can fail in two ways and the user can easily detect both failures. Denote $c_1$ to be the constant as defined above. One type of failure is when KNearestNeighbors returns $\kappa < (1 - \varepsilon c_1 n)k$ sites, in which case the sketching algorithm decodes the $\kappa$ sites correctly. Another type of failure is when KNearestNeighbors returns $\kappa > (1 + \varepsilon c_1 n)k$ sites, in which case the sketching algorithm, designed to decode at most $(1 + \varepsilon)k$ sites, fails to decode the sites
correctly. Of \( b \) independent computations of KNearestNeighbors, the probability that all of them failed is
\[
Pr(b \text{ failures of KNearestNeighbors}) = (Pr(\text{KNearestNeighbors failed}))^b.
\]

8 The System

8.1 \( k \)-Nearest Sites On A Map

We implemented the algorithms in this paper into a system that finds the nearest hotels to a query location in Boston. The system was run on MIT’s OpenStack infrastructure. It maintained a database of 580 hotels in Boston area (given in clear-text). Prior to sending a query to the system a client needs to generate a pair \((P_k, S_k)\) of public key and secret key. To query the system, the client sends an encrypted location \([q]\) and the public key used for encrypting it. For multiple queries, the client can use the same \((P_k, S_k)\) pair and the public key can be uploaded only once to the system. The encryption of \( q \) is done by the client, and the encrypted location can be sent from a browser running on a smart phone or from a quadcopter. The system then runs our secure search algorithm on the cloud using the client’s public key to encrypt constants as needed by the algorithm, and returns an indicator vector of the \( k = 13 \) nearest hotels to \( q \). I.e., we set the auxiliary data to be a single set bit for all hotels. Changing the system to return the name or location of the hosts is easy by changing the auxiliary data. We set \( \varepsilon = \frac{13}{580} \) taken as the error parameter. The list (and the query location) can be decrypted only with the private key available only to the user. The server can update the database of hotels at any time since it is kept in clear-text, for example, by scraping the Internet or negotiating with the hotels themselves.

8.1.1 Hardware

Our system is generic but in this section we evaluate it on MIT’s OpenStack cloud infrastructure. We used one of the standard suggested servers, with 12 2.2 GHz Intel Xeon E5-2630 core and 28 GigaByte of RAM. These cores are common in standard laptops and servers.

8.1.2 Open Software and Security

The algorithms were implemented in C++. HELib library [34] was used for the FHE scheme implementation, including its usage of SIMD (Single Instruction Multiple Data) technique. The source of our system is open under the GNU v3 license and can be found in [34]. Our system and all the experiments below use a security key of 80 bits. This settings can be easily changed by the client.

8.1.3 Experimental Results

In this section we describe the experiments we ran and their results. We ran experiments on a system implementing Algorithm [1]

Data We tested our system on a list of 580 hotels in the Boston area, given by their GPS coordinates. As expected the distribution of hotels around Boston is not even. There are many more hotels around down-town Boston, for example. We drew a grid on the map around Boston and rounded each hotel to the nearest grid point. We repeated our tests with grid of various sizes: \(100 \times 100, 80 \times 80 \) and \(60 \times 60\).

The Experiment We ran Algorithm [1] on 580 hotel locations in Boston area, rounding hotel locations to be on a grid of \( L \times L \) and taking \( p = 2L \). We drew a random query point, \( q \) on the grid and ran KNearestNeighbors\(_{p,n,k,c,R,d,s,aux}(q)\).

For the experiments that had more than 580 points we duplicated the Boston area hotels as many times as needed. Recall that since the server operates on encrypted data, the running time is oblivious to the actual coordinates of the query point \( q \) or the hotels’ coordinates \( s_i \), since the server performs the same operations regardless to the input.
Results  Our experimental results are summarized in Figure 1. The experiments measured the total time to: (i) compute the distances from the query point $q$ to each hotel, $d_i = |q - s_i|$; (ii) compute statistical functions ($\mu$ and $\sigma$) on $d_i$, and (iii) create an indicator vector $\chi$ by comparing each $d_i$ to a threshold computed from the statistical functions. Each of these steps is “embarrassingly parallel” \[36\] which means the running time on each core decreases linearly with the number of cores.

What dominates the slopes of the graphs? Our analysis in Section 7.5 suggests that the time to compute the $k$-nearest neighbors is

$$T = O\left(MUL \cdot \frac{L \log L + n \log L}{CORES \cdot SIMD}\right)$$

where

- $L \times L$ is the grid size to which we rounded the locations of the hotels
- $CORES$ is the number of computation machines (practically, number of core processors that work in parallel)
- $SIMD$ (Single Instruction Multiple Data) is the amount of integers that are packed into a single ciphertext. This $SIMD$ factor is a function of the ring size $p$ and the degree of the evaluated polynomial $O(\log L)$; in HELib, this parameter can be read by calling EncryptedArray::size(), see \[26\].
- $MUL$ is the times for computing a single multiplication operation.

We had $CORES = 10$ in all experiments, and the other parameters did not change once the grid size was set. Our formula, therefore, predicts that the graphs should be linear, as they indeed are. The parameters $L$ and $MUL$ did not change significantly for different grid sizes and the parameter whose value dominated the slope of the (linear) graph was $SIMD = 1288, 1520$ and 980, for grid sizes $60 \times 60$, $80 \times 80$ and $100 \times 100$, respectively. We plotted in dotted lines the linear graph as was predicted by our formula with the various $SIMD$ values.

![Time to perform secure geographic search](image)

Figure 1: Server’s running time (y-axis) on a single machine on MIT’s OpenStack cloud infrastructure, for different database sizes (x-axis) of KNearestNeighbors (Algorithm 1) on encrypted query. Each colored curve represents a different grid size. The dotted lines are the graphs that were expected by our formula.

8.2 Computing Statistical Functions

Since we believe our Algorithm 3 for computing $\frac{1}{m} \sum x_i^c$ is a useful tool in other applications as well, we provide experimental results dedicated to this.
8.2.1 System Overview

For this simple system, we again assumed the user has a pair \((P_k, S_k)\) of public and secret key, such that a message encrypted with \(P_k\) can be decrypted only with \(S_k\). The input for this system were a list of values \(x_1, \ldots, x_n\) encrypted with \(P_k\) by the user. We also assumed \(P_k\) is known to the server. The output of the system was the encrypted value of \(f_e(x_1, \ldots, x_n) = \frac{1}{n} \sum x_i^2\) which can be decrypted only with the secret key \(S_k\). Although this alone is not an interesting functionality for a system, we believe that many systems include a similar computation as a sub system within them. We ran this system as well on MIT’s OpenStack infrastructure.

The Experiment We computed the sums \(\frac{1}{n} \sum x_i\) and \(\frac{1}{n} \sum x_i^2\), i.e. \(e = 1\) (the average) and \(e = 2\) (the average of the squares), the first two moments. We tested with various values of \(1,000 \leq n \leq 100,000\), and ran experiments with \(x_i \in [60]\), \(x_i \in [80]\) and \(x_i \in [100]\). The server is oblivious to the values of \(x_i\) since they are encrypted, and the time of computation is not influenced by the values of \(x_i\).

Results Our experimental results are summarized in Figure 2. In Fig. 2 we show that indeed the formula correctly predicts the experimental results. Assume we are computing \(\frac{1}{m} \sum x_i^e\) for some values of \(n, m\) and \(e\), where \(x_i < L\) for some value of \(L\) and \(1 \leq i \leq n\), then the overall running time is:

\[
T = O \left( \frac{n \cdot \log L \cdot MUL}{CORES \cdot SIMD} \right),
\]

where \(CORES\), \(SIMD\) and \(MUL\) are as explained in section 8.1.3.

![Figure 2: Server’s running time (y-axis) on a single core on MIT’s OpenStack cloud infrastructure, for computing the first two moments (\(\frac{1}{n} \sum x_i\) and \(\frac{1}{n} \sum x_i^2\)) (Algorithm 3) of \(n\) (x-axis) encrypted values. Solid lines are for the time to compute the first moment; dashed lines are for the time to compute the second moment. The dotted lines are the graphs that were expected by our formula. For each moment, the values were taken from three sets: \([0, \ldots, 60]\), \([0, \ldots, 80]\) and \([0, \ldots, 100]\), each represented with a different color.](image)

What dominates the slope of the graph? Although all graphs are linear as expected from our theoretic analysis, it is interesting to note how their slopes differ. From the formula we gave above, the slope of each graph should be proportional to \(\frac{\log L \cdot MUL}{CORES \cdot SIMD}\). In our experiments we had \(CORES = 1\) and negligible differences in \(\log L\). Also the differences we measured in the values of \(MUL\) were negligible. The values of \(SIMD\) were chosen automatically by HELib upon key generation and varied greatly between experiments from 192 to 660. To compare our measured results to theory we plotted with dotted lines the linear functions predicted by our formula: \(f(x) = xc^{SIMD}\), where \(c = \frac{\log L \cdot MUL}{CORES}\) and \(SIMD\) is the SIMD factor that was set in each experiment. It can be seen that the experiments match our expected formula.
References

[1] A. Akavia, D. Feldman, and H. Shaul. Secure database queries on the cloud: Homomorphic encryption meets coresets, 2017. Submitted.

[2] A. Akavia, D. Feldman, and H. Shaul. Secure search on the cloud via coresets and sketches. *arXiv preprint arXiv:1708.05811*, 2017.

[3] A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification with gene expression profiles. *Journal of Computational Biology*, 7(3-4):559–583, July 2004.

[4] J. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. 50, 05 2014.

[5] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS ’12*, pages 309–325, New York, NY, USA, 2012. ACM.

[7] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[8] V. Braverman, G. Frahling, H. Lang, C. Sohler, and L. F. Yang. Clustering high dimensional dynamic data streams. *arXiv preprint arXiv:1706.03887*, 2017.

[9] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku. Streaming k-means on well-clusterable data. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 26–40. Society for Industrial and Applied Mathematics, 2011.

[10] G. S. Çetin, W. Dai, Y. Doröz, W. J. Martin, and B. Sunar. Blind web search: How far are we from a privacy preserving search engine? *IACR Cryptology ePrint Archive*, 2016:801, 2016.

[11] G. S. Çetin, Y. Doröz, B. Sunar, and E. Savas. Depth optimized efficient homomorphic sorting. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 25-26, 2015, Proceedings*, pages 61–80, 2015.

[12] G. S. Çetin, Y. Doröz, B. Sunar, and E. Savas. Low depth circuits for efficient homomorphic sorting. *IACR Cryptology ePrint Archive*, 2015:274, 2015.

[13] A. Chatterjee, M. Kaushal, and I. Sengupta. *Accelerating Sorting of Fully Homomorphic Encrypted Data*, pages 262–273. Springer International Publishing, Cham, 2013.

[14] A. Chatterjee and I. Sengupta. Searching and sorting of fully homomorphic encrypted data on cloud. *IACR Cryptology ePrint Archive*, 2015:981, 2015.

[15] K. L. Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010.

[16] A. Czumaj, C. Lammersen, M. Monemizadeh, and C. Sohler. (1+ ε)-approximation for facility location in data streams. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1710–1728. Society for Industrial and Applied Mathematics, 2013.

[17] Y. Doröz, B. Sunar, and G. Hammouri. Bandwidth efficient pir from ntru. In *Financial Cryptography Workshops*, pages 195–207, 2014.
[18] W. Du, S. Chen, and Y. S. Han. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In Proceedings of the 4th SIAM International Conference on Data Mining, pages 222–233, 2004.

[19] S. Dudoit, J. Fridlyand, and T. P. Speed. Comparison of discrimination methods for the classification of tumors using gene expression data. Journal of the American Statistical Association, 97(457):77–87, 2002.

[20] Y. Gahi, M. Guennoun, and K. El-Khatib. A secure database system using homomorphic encryption schemes. arXiv preprint arXiv:1512.03498, 2015.

[21] C. Gentry. A Fully Homomorphic Encryption Scheme. PhD thesis, Stanford University, Stanford, CA, USA, 2009. AAI3382729.

[22] C. Gentry. Fully homomorphic encryption using ideal lattices. In Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, STOC ’09, pages 169–178, New York, NY, USA, 2009. ACM.

[23] O. Goldreich. Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, New York, NY, USA, 2004.

[24] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC ’87, pages 218–229, New York, NY, USA, 1987. ACM.

[25] S. Halevi. Helib - an implementation of homomorphic encryption. https://github.com/shaih/HElib/, 2013.

[26] S. Halevi and V. Shoup. Algorithms in helib. In 34rd Annual International Cryptology Conference, CRYPTO 2014. Springer Verlag, 2014.

[27] J. Huggins, T. Campbell, and T. Broderick. Coresets for scalable bayesian logistic regression. In Advances in Neural Information Processing Systems, pages 4080–4088, 2016.

[28] P. Indyk, H. Q. Ngo, and A. Rudra. Efficiently decodable non-adaptive group testing. In Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’10, pages 1126–1142, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[29] Y. Lindell and B. Pinkas. Privacy preserving data mining. In Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO ’00, pages 36–54, London, UK, UK, 2000. Springer-Verlag.

[30] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW ’11, pages 113–124, New York, NY, USA, 2011. ACM.

[31] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing, 1973.

[32] J. M. Phillips. Coresets and sketches. arXiv preprint arXiv:1601.00617, 2016.

[33] S. S. Roy, F. Vercauteren, J. Vliegen, and I. Verbauwhede. Hardware assisted fully homomorphic function evaluation and encrypted search. IEEE Transactions on Computers, 2017.

[34] H. Shaul. LiPHE: Library for practical homomorphic encryption., 2017.

[35] J. Vaidya, M. Kantarcıoğlu, and C. Clifton. Privacy-preserving naïve bayes classification. The VLDB Journal, 17(4):879–898, July 2008.
[36] B. Wilkinson and M. Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, 1999.

[37] D. P. Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[38] A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS ’86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.