Parallel Homomorphic Encryption

Seny Kamara
Microsoft Research
senyk@microsoft.com

Mariana Raykova [*]
Columbia University
mariana@cs.columbia.edu

Abstract

In the problem of private outsourced computation, a client wishes to delegate the evaluation of a function f on a private input x to an untrusted worker without the latter learning anything about x and $f(x)$. This problem occurs in many applications and, most notably, in the setting of cloud computing.

In this work, we consider the problem of privately outsourcing computation to a *cluster* of machines, which typically happens when the computation is to be performed over massive datasets, e.g., to analyze large social networks or train machine learning algorithms on large corpora. At such scales, computation is beyond the capabilities of any single machine so it is performed by large-scale clusters of workers.

We address this problem by introducing the notion of *parallel* homomorphic encryption (PHE) schemes, which are encryption schemes that support computation over encrypted data via evaluation algorithms that can be efficiently executed in parallel. We also consider *delegated* PHE schemes which, in addition, can hide the function being evaluated. More concretely, we focus on the MapReduce model of parallel computation and show how to construct PHE schemes that can support various MapReduce operations on encrypted datasets including element testing and keyword search. More generally, we construct schemes that can support the evaluation of functions in NC^0 with locality 1 and $m = \mathsf{polylog}(k)$ (where k is the security parameter).

Underlying our PHE schemes are two new constructions of (local) randomized reductions (Beaver and Feigenbaum, *STACS* '90) for univariate and multivariate polynomials. Unlike previous constructions, our reductions are not based on secret sharing and are *fully-hiding* in the sense that the privacy of the input is guaranteed even if the adversary sees *all* the client's queries.

Our randomized reduction for univariate polynomials is information-theoretically secure and is based on permutation polynomials, whereas our reduction for multivariate polynomials is computationally-secure under the multi-dimensional noisy curve reconstruction assumption (Ishai, Kushilevitz, Ostrovsky, Sahai, *FOCS '06*).

1 Introduction

In the problem of private outsourced computation, a client wishes to delegate the evaluation of a function f on a private input x to an untrusted worker without the latter learning anything about x and $f(x)$. This problem occurs in many applications and, most notably, in the setting of cloud computing, where a provider makes its computational resources available to clients "as a service".

One approach to this problem is via the use of homomorphic encryption (HE). An encryption scheme is homomorphic if it supports computation on encrypted data, i.e., in addition to the standard encryption

[*]Work done at Microsoft Research.

and decryption algorithms it also has an evaluation algorithm that takes as input an encryption of some message x and a function f and returns an encryption of $f(x)$. HE schemes can be roughly categorized into two types. The first are *arithmetic* HE schemes which, in addition to the standard encrypt and decrypt operations, have an add or multiply operation that take as inputs encryptions of messages x_1 and x_2 and returns encryptions of $x_1 + x_2$ and $x_1 \cdot x_2$, respectively. If an arithmetic HE scheme supports both addition and multiplication, then it can evaluate any arithmetic circuit over encrypted data and we say that it is a fully homomorphic encryption (FHE) scheme. We refer to the second type of HE schemes as non-arithmetic since they do not provide (at least explicitly) addition or multiplication operations.

The problem of outsourced computation occurs in various forms. For instance, in addition to the simple client/worker setting described above, clients often wish to outsource their computation to *clusters* of workers. This typically occurs when the computation is to be performed over massive datasets, e.g., to analyze large social networks or train machine learning algorithms on large corpora. At such scales, computation is beyond the capabilities of any single machine so it is performed on clusters of machines, i.e., large-scale distributed systems often composed of low-cost unreliable commodity hardware. For our purposes, we will view such a cluster as a system composed of w workers and one controller. Given some input, the controller generates n jobs (where typically $n \gg w$) which it distributes to the workers. Each worker executes its job in parallel and returns some value to the controller who then decides whether to continue the computation or halt.

In this work, we consider the problem of privately outsourcing computation to a *cluster* of machines. To address this, we introduce *parallel* homomorphic encryption (PHE) schemes, which are encryption schemes that support computation over encrypted data through the use of an evaluation algorithm that can be efficiently executed in parallel. Using a PHE scheme, a client can outsource the evaluation of a function f on some private input x to a cluster of w machines as follows. The client encrypts x and sends the ciphertext and f to the controller. Using the ciphertext, the controller generates n jobs that it distributes to the workers and, as above, the workers execute their jobs in parallel. When the entire computation is finished, the client receives a ciphertext which it decrypts to recover $f(x)$. To handle cases where even the function f must be hidden, we introduce a second variant of PHE which we refer to as *delegated* PHE. This variant includes an additional token generation algorithm that takes as input f and outputs a token τ that reveals no information about f but that, nonetheless, can be used by the evaluation algorithm (instead of f) to return an encryption of $f(x)$.

Applications of PHE. As discussed above, the most immediate application of PHE is to the setting of outsourced computation where a weak computational device wishes to make use of the resources of a more powerful server. Clearly, to be useful in this setting it is crucial that either: (1) running the encryption and decryption operations of the PHE scheme take less time than evaluating f on the input x directly; or (2) the PHE scheme is *multi-use* in the sense that the evaluations of several (different) functions can be done on a single ciphertext (this is also referred to as the online/offline setting). In this work we focus on the latter and present several multi-use PHE schemes. Using our schemes a client can encrypt a large database during an offline phase and then, have the workers evaluate many different functions on its data during the online phase. In particular, the client does not need to know the functions it will want to evaluate during the online phase at the time of encryption. In section 9 we show how our PHE schemes can be used to evaluate several functionalities (e.g., keyword search, set membership testing, and disjunctions) for a particular model of parallel computation which we describe below.

Parallel computation. Most computations are not completely parallelizable and require some amount of communication between machines. The specifics of how the computation and communication between processors are organized leads to particular architectures, each having unique characteristics in terms of computational and communication complexity. This has motivated the design of several architecture-independent models of parallel computation, including NC circuits [16], the parallel RAM (PRAM) [25, 40], Valiant's bulk synchronous parallel (BSP) model [51], LogP [20] and, more recently, the MapReduce [23] and Dryad models [33]. It follows that an important consideration in the design of PHE schemes is the parallel model in which the function will be evaluated. In this work, we focus on the MapReduce model (which we describe below) but note that our choice is due mainly to practical considerations (e.g., the emergence of cloud-based MapReduce services such as Amazon's Elastic MapReduce [1]) and that PHE can also be considered with respect to other models of parallel computation. As an example, note that any arithmetic FHE scheme yields an NC-parallel HE scheme for any function f in NC.

Applications to cloud-based cluster-computing. Due to its simplicity and generality, the MapReduce model has quickly become the standard for working with massive datasets. In fact, it is arguably the most successful and widely used model of parallel computation. In 2008 it was reported that Google processed over 20 petabytes of data a day using MapReduce [24] and that Yahoo! deployed a $10,000$ core MapReduce cluster [4]. A MapReduce algorithm is run by an execution framework that handles the details of distributing work among the machines in the cluster, balancing the workload so as to optimize performance and recovering from failures. The most popular framework is Hadoop [3] which is open source and used by hundreds of large organizations including Amazon, Ebay, Facebook, Yahoo!, Twitter and IBM.

Building and maintaining large-scale clusters requires a considerable amount of effort and resources, so a recent trend in cluster-computing has been to make use of cloud infrastructures. Examples include Amazon's Elastic MapReduce [1], Cloudera's Hadoop distribution [2] (which can run over several cloud infrastructures) and the recently announced Microsoft Azure Hadoop service. With such services, a client can run a MapReduce algorithm on massive datasets "in the cloud". While these services allow clients to take advantage of all the benefits of cloud computing, they require the client to trust the provider with its data. For many potential clients (e.g., hospitals or government agencies) this presents an unacceptable risk.

Using an MR-parallel HE scheme a client can maintain the confidentiality of its data while utilizing the processing power of a third-party MapReduce cluster such as Amazon's Elastic MapReduce service. Of course, the client must bear the costs of encryption and decryption which, for massive datasets, can represent a non-trivial amount of work. But, as discussed above, in certain settings this cost is dominated by the amount of work that is outsourced. This occurs, for example, if the function being evaluated is very complex or if the client wishes to evaluate many different functions over its data (i.e., the offline/online setting). We also note that for all of our constructions, encryption can be performed in a streaming manner. This means that even if the data is very large, it can still be encrypted by a "weak" client (i.e., with a small amount of memory) albeit rather slowly.

1.1 Overview of Techniques

Designing PHE schemes. Our approach to designing PHE schemes combines randomized reductions (RR) [12, 13] and homomorphic encryption. Roughly speaking, a RR from a function f to a function g transforms an input x in the domain of f to a set of n inputs $S = (s_1, \ldots, s_n)$ in the domain of g such that $f(x)$ can be efficiently recovered from $(g(s_1), \ldots, g(s_n))$. In addition, a RR guarantees that no information about x or $f(x)$ can be recovered from any subset of $t \leq n$ elements of S. A natural approach to constructing a PHE (ignoring the particualr model of parallel computatin) is therefore to "encrypt" x by using a RR to transform it into a set $(s_1, \ldots, s_n)$ and have each worker i evaluate g on s_i independently. The results can then be sent back to the client who can recover $f(x)$ using the reduction's recover algorithm. As long as at most t workers collude, the RR will guarantee the confidentiality of x and $f(x)$.

Unfortunately, there are two problems with this approach. First, as far as we know, the best hiding threshold achieved by any RR is $t \leq (n-1)/q$ [12, 13]. In the context of cloud computing, however, this is not a reasonable assumption as the cloud provider owns *all* the machines in the cluster [1]. Another limitation is that the client has to run the RR's recovery algorithm which can represent a non-trivial amount of work depending on the particular scheme and the parameters used.

We address these limitations in the following way. First, we show how to construct *fully-hiding* RRs, i.e., reductions with a hiding threshold of $t = n$. Our first construction is for the class of univariate polynomials while the second is for multivariate polynomials with a "small" (i.e., poly-logarithmic in the security parameter) number of variables. As far as we know, these are the first RRs to achieve a threshold of $t = n$. Towards handling the second limitation, we observe that if the recovery algorithm of the RR can be evaluated homomorphically, then the recovery step can also be outsourced to the workers. Clearly, using FHE any recovery algorithm can be outsourced, but our goal here is to avoid the use of FHE so as to have practical schemes. Our approach therefore will be to design RRs with recovery algorithms that are either (1) simple enough to be evaluated using partially-homomorphic encryption; or (2) efficient enough to be run by the client.

Designing fully-hiding RRs. The best known RRs for polynomials [12, 13] work roughly as follows. Let $\mathbf{Q}$ be the polynomial of degree q that we wish to evaluate and $\mathbf{x} \in \mathbb{F}^m$ be the input. First, each element of $\mathbf{x}$ is shared into $q \cdot t + 1$ shares using Shamir secret sharing with a sharing polynomial of degree t (i.e., the hiding threshold). This yields m sets of shares $(\mathbf{s}_1, \ldots, \mathbf{s}_m)$, where $\mathbf{s}_i = (\mathbf{s}_i[1], \ldots, \mathbf{s}_i[q \cdot t + 1])$. Each worker $j \in [q \cdot t + 1]$ is then given $(\mathbf{s}_1[j], \ldots, \mathbf{s}_m[j])$ and evaluates $\mathbf{Q}$ on his shares. Given the results of all these evaluations, the client interpolates at 0 to recover $\mathbf{Q}(\mathbf{x})$. This approach yields a hiding threshold of up to $t = (n-1)/q$. Note that this construction works equally as well for $m = 1$. As shown in [12, 13], this can be improved to $t = n \cdot c \log(m)/m$ for any constant $c > 0$ and $m > 1$.

Due to their reliance on secret sharing, it is not clear how to extend the techniques from [12, 13] to achieve $t = n$ and (informally) it seems hard to imagine using any technique based on secret sharing to achieve full hiding. Instead, we introduce two new techniques for designing RRs. The first works for univariate polynomials and makes use of permutation polynomials over finite fields (i.e., bijective families of polynomials). The resulting RR is information-theoretically secure and very efficient. Our second approach is only computationally-secure but works for multivariate polynomials. The security of the reduction is based on the multi-dimensional noisy curve reconstruction assumption [36, 45].

1.2 Our Contributions

While (sequential) homomorphic encryption constitutes an important step towards private outsourced computation, an increasing fraction of the computations performed "in the cloud" is on massive datasets and therefore requires the computation to be performed on clusters of machines. To address this, we make the following contributions:

[1]Of course one could use the above approach with more than one cloud providers if they do not collude.

1. We initiate the study of PHE and delegated PHE (which also hides the function being evaluated). In particular, we consider the MapReduce model of parallel computation and formalize MapReduce-parallel HE schemes. Given the practical importance of the MapReduce model and the emergence of cloud-based MapReduce clusters, we believe the study of MapReduce-parallel HE to be important and well motivated.

2. We construct new RRs for univariate and multivariate polynomials with a small number of variables (i.e., polylogarithmic in the security parameter). Our reduction for univariate polynomials is information theoretically secure while our reduction for multivariate polynomials is secure based on the multi-dimensional noisy curve reconstruction assumption [36]. Both our constructions achieve a hiding threshold of $t = n$ and are, as far as we know, the first constructions to do so.

3. We give a general transformation from any RR to a MR-parallel HE scheme given any public-key HE scheme that can evaluate the reductions' recovery algorithm. If the RR works for any function within a class $\mathcal{C}$, then the resulting MR-parallel scheme is $\mathcal{C}$-homomorphic.

4. We show how to construct a *delegated* MR-parallel HE scheme for any function whose output values can be computed by evaluating a (single) univariate polynomial over the input values. Our construction is based on our RR for univariate polynomials and makes black-box use of a 2DNF-HE scheme (i.e., a HE scheme that can handle a single multiplication and any number of additions) such as [15, 31].

5. We show how to construct a MR-parallel HE scheme for any function in NC^0_m which consists of all functions whose outputs depend on at most m input elements. We note that the construction is *not* delegated. The scheme makes use of our RR for multivariate polynomials and makes black-box use of additively HE.

6. Finally, we show how our MR-parallel HE schemes can be used to evaluate various database queries on encrypted datasets including keyword search and set membership testing.

2 Related Work

We already mentioned previous work on parallel computation so we restrict the following discussion to work closer to our own.

Randomized reductions & encodings. Private outsourced computation is also known as *instance hiding* and was first considered by Abadi, Feigenbaum and Kilian [5]. Abadi et al. show that if a function f is NP-hard then it cannot be privately outsourced (even using interaction) unless the polynomial hierarchy collapses. This negative result motivated Beaver and Feigenbaum [11] to consider a setting where the client outsources to multiple servers that are not allowed to communicate. Under these constraints, the authors show that every function f can be privately outsourced by constructing a RR for multi-variate polynomials that achieves a hiding threshold of $t = 1$ (i.e., without any resistance to collusion). Following [11], Beaver, Feigenbaum, Killian and Rogaway showed how to achieve $t = (n-1)/q$ for any m-variate polynomial with $m \geq 1$ and $t = n \cdot c \log(m)/m$ for $c, m > 1$ [12, 13].

A related notion to RRs are randomized encodings (RE), first proposed by Ishai and Kushilevitz for the purpose of constant-round multi-party computation [34, 35] and later used for the construction of various cryptographic primitives (e.g., pseudo-random generators, commitment schemes, public-key encryption) with efficient parallel complexity [7, 6, 8, 9]. While REs imply RRs the converse is not known to be true [10].

Homomorphic encryption. As mentioned in section 1, the problem of private outsourced computation can be addressed non-interactively using HE. Of course, several semantically secure arithmetic HE schemes are known [32, 27, 14, 46, 44, 47, 21, 15, 31], including Gentry's breakthrough FHE scheme [29] and its variants [52, 50]. Several non-arithmetic HE schemes are also known including [49, 17, 37, 30].

MapReduce. MapReduce was introduced by Dean and Ghemawat in [23]. In this work, they describe the design and implementation of the MapReduce framework for processing massive datasets on large-scale systems of loosely coupled machines. MapReduce offers a simple interface to design and implement parallel algorithms and an execution framework that handles the details of distributing work among the machines in the cluster, balancing the workload so as to optimize performance and recovering from failures. Due to the simplicity and generality of the MapReduce model, it has quickly become the standard for working with massive datasets

The MapReduce model of computation has been formalized and studied in recent work by Karloff, Suri and Vassilvitskii [39]. This work introduces a new complexity class that captures the power of the model and relates it to known models of parallel computation, including PRAMs and NC circuits. The authors also present new MapReduce algorithms for frequency counts, undirected s-t connectivity and for computing the minimum spanning tree of a dense graph.

Other MapReduce algorithms have been proposed for a multitude of tasks, including for indexing the Web and computing PageRank [23], clustering high-dimensional data [22], processing large-scale graphs [38, 18, 43, 39], machine learning [19] and natural language processing [48].

3 Preliminaries and Notation

Notation. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \overset{\$}{\leftarrow} X$ to represent an element x being sampled uniformly from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. We refer to the ith element of a vector $\mathbf{v}$ as either v_i or $\mathbf{v}[i]$. Throughout k will refer to the security parameter. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every polynomial $p(\cdot)$ and sufficiently large k, $\nu(k) < 1/p(k)$. Let $\mathsf{poly}(k)$ and $\mathsf{negl}(k)$ denote unspecified polynomial and negligible functions in k, respectively. We write $f(k) = \mathsf{poly}(k)$ to mean that there exists a polynomial $p(\cdot)$ such that for all sufficiently large k, $f(k) \leq p(k)$, and $f(k) = \mathsf{negl}(k)$ to mean that there exists a negligible function $\nu(\cdot)$ such that for all sufficiently large k, $f(k) \leq \nu(k)$. The statistical distance between two distributions χ_1 and χ_2 over the same space X is defined as $\mathsf{SD}(\chi_1, \chi_2) = \max_{S \subset X} |\Pr[\, \chi_1 \in S\,] - \Pr[\, \chi_2 \in S\,]|$.

Polynomials. If p is a univariate polynomial of degree d over a field $\mathbb{F}$, then it can be written as $p(x) = \sum_{\alpha \in S} p(\alpha) \cdot \mathcal{L}_\alpha(x)$, where S is an arbitrary subset of $\mathbb{F}$ of size $d+1$ and $\mathcal{L}_\alpha$ is the Lagrangian coefficient defined as $\mathcal{L}_\alpha(x) = \prod_{i \in S, i \neq \alpha}(x-i)/(\alpha-i)$. A permutation polynomial $p \in \mathbb{F}[x]$ is a bijection over $\mathbb{F}$. One class of permutation polynomials which will make use of in this work are the Dickson polynomials (of the first kind) which are a family of polynomials $\mathbf{D} = \{\mathbf{D}_{d,\beta}\}$ over a finite field $\mathbb{F}$ indexed by a degree $d > 0$ and a non-zero element $\beta \in \mathbb{F}$. If $|\mathbb{F}|^2 - 1$ is relatively prime to d and if $\beta \neq 0$, then the Dicskon polynomial $\mathbf{D}_{d,\beta}$ defined as

$$\mathbf{D}_{d,\beta}(x) \overset{def}{=} \mathbf{D}_d(x,\beta) = \sum_{\lambda=0}^{\lfloor d/2 \rfloor} \frac{d}{d-\lambda} \cdot \binom{d-\lambda}{\lambda} \cdot (-\beta)^{\lambda} x^{d-2\lambda},$$

is a permutation over $\mathbb{F}$. For $d = 2$ and any $\beta \neq 0$, we have $\mathbf{D}_{2,\beta}(x) = x^2 - 2\beta$ which is a permutation over any $\mathbb{F}$ such that $|\mathbb{F}|^2 - 1$ is odd.

Locality and degree. Here, we follow closely the notation and definitions of [7]. We view a function $f : \mathbb{F}^* \to \mathbb{F}^*$ as a function family $\{f_n\}_{n\in\mathbb{N}}$ where f_n is the restriction of f to inputs of length n. Throughout this work we will represent a function $f_n : \mathbb{F}^n \to \mathbb{F}^w$ as a vector of polynomials $(\mathbf{Q}_1, \ldots, \mathbf{Q}_w)$ such that the ith output of f_n can be computed using $\mathbf{Q}_i$. We assume that f can be computed in polynomial time in n and that its vector representation can be found in polynomial time.

We say that f is ℓ-local if all its output elements depend on at most ℓ input elements and that it has degree d if each polynomial in its vector representation has degree at most d. The (arithmetic) class NC^0_ℓ includes all ℓ-local functions that are computable and constructible in polynomial time (throughout this work all complexity classes will be uniform). Furthermore, we denote by $\mathrm{NC}^0_{\ell,\mathsf{eq}}$ the class of functions that are ℓ-local and whose vector representation is such that $\mathbf{Q}_1 = \cdots = \mathbf{Q}_w$.

Homomorphic encryption. Let $\mathcal{F}$ be a family of of n-ary functions. A $\mathcal{F}$-homomorphic encryption scheme is a set of four polynomial-time algorithms $\mathsf{HE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ such that Gen is a probabilistic algorithm that takes as input a security parameter k and outputs a secret key K; Enc is a probabilistic algorithm that takes as input a key K and an n-bit message m and outputs a ciphertext c; Eval is a (possibly probabilistic) algorithm that takes as input a function $f \in \mathcal{F}$ and n encryptions $(c_1, \ldots, c_n)$ of messages $(m_1, \ldots, m_n)$ and outputs an encryption c of $f(m_1, \ldots, m_n)$; and Dec is a deterministic algorithm takes as input a key K and a ciphertext c and outputs a message m. In this work, we make use of 2DNF-HE schemes which support an arbitrary number of additions and a single multiplication. Concrete instantiations of such schemes include [15] and [31]. Informally, a HE scheme is secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not leak any useful information about the plaintext even to an adversary that (in the case of a private-key scheme) can adaptively query an encryption oracle. As discussed in [28] (cf., section 2.2.), the formal definition of CPA-security for standard encryption schemes can be used for HE schemes.

4 MapReduce-Parallel Homomorphic Encryption

In this section, we first give an overview of the MapReduce model of computation together with an example of a simple MapReduce algorithm. We refer the reader to [42] for a more detailed exposition. After formalizing the MapReduce model, we define MapReduce-parallel HE schemes and present our security definitions for standard and delegated MR-parallel HE schemes.

4.1 The MapReduce Model of Computation

At a high level, MapReduce works by applying a map operation to the data which results in a set of label/value pairs. The map operation is applied in *parallel* and the resulting pairs are routed to a set of "reducers". All pairs with the same label are routed to the same reducer which is then tasked with applying a reduce operation that combines the values into a single value for that label.

More precisely, a MapReduce algorithm is divided into five (possibly probabilistic) algorithms: *parse*, *map*, *partition*, *reduce* and *merge*. The parse algorithm takes an n-bit input x and outputs a set of label/value pairs $\{(\ell_i, v_i)\}_i$. The map algorithm takes a single *input pair* (ℓ_i, v_i) and outputs a set of *intermediate* pairs $\{(\lambda_j, \gamma_j)\}_j$, where λ_j is from some label space Λ. Note that the intermediate pairs do not have to be from the same space as the input pairs. The partition algorithm takes as input a set of intermediate pairs and partitions it into the sets $\{P_l\}_l$. The reduce algorithm takes as input a label λ and a partition P_l and returns a string z. Finally, the merge algorithm takes a set of strings $\{z_r\}_r$ and returns an output y. For the algorithm to work properly, it is crucial that the map and reduce algorithms be stateless.

Definition 4.1 (MapReduce algorithm)**.** *A MapReduce algorithm for a function family $\mathcal{F}$ is a tuple of five polynomial-time algorithms $\Pi = (\mathsf{Parse}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge})$ such that:*

> $(\ell_i, v_i)_i \leftarrow \mathsf{Parse}(f, x)$*: is a deterministic algorithm that takes as input a function $f \in \mathcal{F}$ and a string x, and that returns a sequence of input pairs.*
>
> $(\lambda_j, \gamma_j)_j \leftarrow \mathsf{Map}(\ell, v)$*: is a (possibly probabilistic) algorithm that takes an input pair (ℓ, v) and that returns a sequence of intermediate pairs.*
>
> $h \leftarrow \mathsf{Part}(\lambda, \gamma)$*: is a (possibly probabilistic) algorithm that takes as input an intermediate pair (λ, γ) and that returns a value h in some space H.*
>
> $(\lambda, z) \leftarrow \mathsf{Red}(\lambda, P)$*: is a (possibly probabilistic) algorithm that takes as input a label λ and a set P of intermediate values and returns an output pair (λ, z).*
>
> $y \leftarrow \mathsf{Merge}\big((\lambda_t, z_t)_t\big)$*: is a deterministic algorithm that takes as input a set of output pairs and returns a string y.*

We will sometimes denote the execution of a MapReduce algorithm Π on a function f and input x as $y \leftarrow \Pi(f, x)$.

We add that all implementations of the MapReduce framework offer a default Part algorithm that maps all the intermediate pairs with the same label to the same partition. This algorithm is usually referred to as the Shuffle algorithm.

Executing a MapReduce Algorithm. A MapReduce algorithm $\Pi = (\mathsf{Parse}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge})$ is executed on a cluster of w workers and one controller as follows (we refer the reader to [23] for a more detailed description). The client provides a function f and an input x to the controller who runs Parse on (f, x), resulting in a sequence of input pairs $(\ell_i, v_i)_i$. Each pair is then assigned by the controller to a worker that evaluates Map on it. Note that since the Map algorithm is stateless, it can be executed in parallel. Typically the number of input pairs is much larger than the number of workers so this stage may require several rounds. When all the input pairs have been processed, the Part algorithm is executed on the set of intermediate pairs. This results in a partition of the intermediate pairs and each element of the partition is then assigned to a worker that runs the Red algorithm. Again, since Red is stateless it can be executed in parallel (though it can be sequential on its own partition). The outputs of all these Red executions are then processed using Merge and the final result is returned to the client. At any time, a worker is either executing the Map algorithm (in which case it is a *mapper*) or the Red algorithm (in which case it is a *reducer*).

An example. Perhaps the simplest example of a MapReduce algorithm is to determine frequency counts, i.e., the number times a keyword occurs in a document collection. The parse algorithm takes the document collection $(D_1, \ldots, D_n)$ as input and outputs a set of input pairs $(i, D_i)_i$. Each mapper receives an input pair (i, D_i) and outputs a set of intermediate pairs $(w_j, 1)_j$ for each word w_j found in D_i. All the intermediate pairs are then partitioned by the partition operation into sets $\{P_l\}$, where P_l consists of all the intermediate pairs with label w_l. The reducers receive a set P_l of intermediate pairs and sum the values of each pair. The result is a count of the number of times the word w_l occurs in the document collection. The merge algorithm then concatenates all these counts and returns the result.

4.2 Syntax and Security Definitions

A MR-parallel HE scheme is a HE whose evaluation operation can be computed using a MapReduce algorithm. We formalize this in the following definition.

Definition 4.2 (MR-parallel HE). *A private-key MR-parallel $\mathcal{F}$-homomorphic encryption scheme is a tuple of polynomial-time algorithms* $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$*, where* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *are as in a private-key encryption scheme and* $\mathsf{Eval} = (\mathsf{Parse}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge})$ *is a MapReduce algorithm. More precisely we have:*

> $K \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic algorithm that takes as input a security parameter k and that returns a key K.*
>
> $c \leftarrow \mathsf{Enc}(K, x)$*: is a probabilistic algorithm that takes as input a key K and an input x from some message space* X*, and that returns a ciphertext c. We sometimes write this as* $c \leftarrow \mathsf{Enc}_K(x)$.
>
> $(\ell_i, v_i)_i \leftarrow \mathsf{Parse}(f, c)$*: is a deterministic algorithm that takes as input a function $f \in \mathcal{F}$ and a ciphertext c, and that returns a sequence of input pairs.*
>
> $(\lambda_j, \gamma_j)_j \leftarrow \mathsf{Map}(\ell, v)$*: is a (possibly probabilistic) algorithm that takes an input pair (ℓ, v) and that returns a sequence of intermediate pairs.*
>
> $h \leftarrow \mathsf{Part}(\lambda, \gamma)$*: is a (possibly probabilistic) algorithm that takes as input an intermediate pair (λ, γ) and that returns a value h in some space H.*
>
> $(\lambda, z) \leftarrow \mathsf{Red}(\lambda, P)$*: is a (possibly probabilistic) algorithm that takes a label λ and a partition P of intermediate values and returns an output pair (λ, z).*
>
> $c' \leftarrow \mathsf{Merge}\big((\lambda_t, z_t)_t\big)$*: is a deterministic algorithm that takes as input a set of output pairs and returns a ciphertext c'.*
>
> $y \leftarrow \mathsf{Dec}(K, c')$*: is a deterministic algorithm that takes a key K and a ciphertext c' and that returns an output y. We sometimes write this as* $y \leftarrow \mathsf{Dec}_K(c')$.

We say that PHE *is correct if for all $k \in \mathbb{N}$, for all $f \in \mathcal{F}_k$, for all K output by* $\mathsf{Gen}(1^k)$*, for all $x \in$* X*, for all c output by* $\mathsf{Enc}_K(x)$, $\mathsf{Dec}_K\big(\mathsf{Eval}(f, c)\big) = f(x)$.

To be usable in the setting of private outsourced computation, a PHE scheme should guarantee that its ciphertexts reveal no useful information about the input x or the output $f(x)$. We note that in this setting it is sufficient for this to hold with respect to a *single* input. In the context of outsourced computation, as opposed that of secure communication, the cost of generating a new key per input is negligible. As such, our security definitions only guarantee security for a single input (which could be, e.g., a massive dataset).

Another issue to consider in the context of HE is the possibility *side-channels* during the evaluation. Such leakage can occur if some aspect of the evaluation of f is data-dependent (e.g., the amount of space used or the time needed to complete) so some care has to be taken to make the evaluation "leakage-resilient". In the case of arithmetic FHE, side-channels are not a concern since evaluation is done by executing a circuit which is a data-oblivious operation. This is not the case for MR-parallel HE, however, since many "natural" MapReduce algorithms appear to be very data-dependent. Consider, for instance, the simple frequency count algorithm described above. The number of intermediate pairs immediately reveals the number of words in a document and the sizes of the partition elements reveal the counts for all the words.

Definition 4.3 (CPA^1-security)**.** *Let* $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge}, \mathsf{Dec})$ *be a MR-parallel* $\mathcal{F}$*-homomorphic encryption scheme and consider the following probabilistic experiments where* $\mathcal{A}$ *is an adversary and* $\mathcal{S}$ *is a simulator:*

$\mathbf{Real}_{\mathsf{PHE},\mathcal{A}}(k)$*: the challenger begins by running* $\mathsf{Gen}(1^k)$ *to generate a key* K*.* $\mathcal{A}$ *outputs an input* x *and receives a ciphertext* $c \leftarrow \mathsf{Enc}_K(x)$ *from the challenger.* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}_{\mathsf{PHE},\mathcal{A},\mathcal{S}}(k)$*:* $\mathcal{A}$ *outputs an input* x*. Given* $|x|$*,* $\mathcal{S}$ *generates and returns a ciphertext* c *to* $\mathcal{A}$*.* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

We say that PHE *is secure against a single-message chosen-plaintext attack if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that*

$$|\Pr[\,\mathbf{Real}_{\mathsf{PHE},\mathcal{A}}(k) = 1\,] - \Pr[\,\mathbf{Ideal}_{\mathsf{PHE},\mathcal{A},\mathcal{S}}(k) = 1\,]| \leq \mathsf{negl}(k),$$

where the probabilities are over the coins of Enc*,* $\mathcal{A}$ *and* $\mathcal{S}$*.*

A delegated PHE scheme is a PHE scheme that supports computation on encrypted data in such a way that the function being computed is hidden. We provide a formal definition below.

Definition 4.4 (Delegated MR-parallel HE)**.** *A delegated MR-parallel* $\mathcal{F}$*-homomorphic encryption scheme is a tuple of nine polynomial-time algorithms* $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Parse}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge}, \mathsf{Dec})$ *such that* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge}, \mathsf{Dec})$ *are as in Definition 4.2 and that* Token *and* Parse *are as follows:*

$\tau \leftarrow \mathsf{Token}(K, f)$*: is a probabilistic algorithm that takes as input a key* K *and a function* f *and that returns a token* τ*. We can write this as* $\tau \leftarrow \mathsf{Token}_K(f)$*.*

$(\ell_i, v_i)_i \leftarrow \mathsf{Parse}(\tau, c)$*: is a deterministic algorithm that takes as input a token* τ *and a ciphertext* c*, and that returns a sequence of input pairs.*

We say that PHE *is correct if for all* $k \in \mathbb{N}$*, for all* $f \in \mathcal{F}_k$*, for all* K *output by* $\mathsf{Gen}(1^k)$*, for all* $x \in \mathsf{X}$*, for all* c *output by* $\mathsf{Enc}_K(x)$*, for all tokens* τ *output by* $\mathsf{Token}(K, f)$*,* $\mathsf{Dec}_K\big(\Pi(\tau, c)\big) = f(x)$*, where* $\Pi = (\mathsf{Parse}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge})$*.*

In addition to CPA^1-security, a delegated PHE scheme should also hide the function being evaluated. This is formally captured in the following definition.

Definition 4.5 (CPFA[1]-security)**.** *Let* $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge}, \mathsf{Dec})$ *be a delegated MR-parallel* $\mathcal{F}$*-homomorphic encryption scheme and consider the following probabilistic experiments where* $\mathcal{A}$ *is an adversary and* $\mathcal{S}$ *is a simulator:*

$\mathbf{Real}^{\star}_{\mathsf{PHE},\mathcal{A}}(k)$: *the challenger begins by running* $\mathsf{Gen}(1^k)$ *to generate a key* K*.* $\mathcal{A}$ *outputs an input* x *and receives a ciphertext* $c \leftarrow \mathsf{Enc}_K(x)$ *from the challenger. The adversary chooses a polynomial number of functions* $f \in \mathcal{F}$ *and, for each* f*, receives a token* $\tau \leftarrow \mathsf{Token}_K(f)$ *from the challenger. Note that each choice of* f *is done adaptively, i.e., as a function of* c*, the previously chosen functions and the previously received tokens. Finally,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}^{\star}_{\mathsf{PHE},\mathcal{A},\mathcal{S}}(k)$: $\mathcal{A}$ *outputs an input* x*. Given* $|x|$*,* $\mathcal{S}$ *generates and returns a ciphertext* c *to* $\mathcal{A}$*. The adversary chooses a polynomial number of functions* $f \in \mathcal{F}$ *and, for each function* f*, the simulator is given* $\deg(f)$*. The simulator returns a token* τ*. Again,* $\mathcal{A}$ *chooses the functions adaptively. Finally,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

We say that PHE *is secure against a single-message chosen-plaintext and function attacks if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that*

$$\left|\Pr\left[\,\mathbf{Real}^{\star}_{\mathsf{PHE},\mathcal{A}}(k) = 1\,\right] - \Pr\left[\,\mathbf{Ideal}^{\star}_{\mathsf{PHE},\mathcal{A},\mathcal{S}}(k) = 1\,\right]\right| \leq \mathsf{negl}(k),$$

where the probabilities are over the coins of Enc, Token, $\mathcal{A}$ *and* $\mathcal{S}$*.*

5 Randomized Reductions for Polynomials

In this section, we formally define randomized reductions [11, 12, 13] and then present our fully-hiding constructions for univariate and multivariate polynomials. Our definitions follow closely the ones given by Beaver, Feigenbaum, Killian and Rogaway [13].

Let $t, n \in \mathbb{N}$ such that $t \leq n$. A function $f : X \to Y$ is (t,n)-locally random reducible to a function $g : \widetilde{X} \to \widetilde{Y}$ if there exists two polynomial-time algorithms $\mathsf{RR} = (\mathsf{Scatter}, \mathsf{Recon})$ that work as follows. $\mathsf{Scatter}$ is a probabilistic algorithm that takes as input an element $x \in X$ and a parameter $n \in \mathbb{N}$, and returns a sequence $\mathbf{s} \in \widetilde{X}^n$ and some state information st. Recon is a deterministic algorithm that takes as input some state st and a sequence $\mathbf{y} \in \widetilde{Y}^n$ and returns an element $y \in Y$. In addition, we require that RR satisfy the following properties:

- (Correctness) for all $x \in X$,

$$\Pr\left[\,\mathsf{Recon}\big(st, g(s_1), \ldots, g(s_n)\big) = f(x) : (\mathbf{s}, st) \leftarrow \mathsf{Scatter}(x, n)\,\right] \geq 3/4,$$

  where the probability is over the coins of $\mathsf{Scatter}$. We depart slightly from the original definition [13] in that here Recon does not need to take x as input.

- (t-hiding) for all $I \subseteq [n]$ such that $|I| = t$, and all x_1 and x_2 in X such that $|x_1| = |x_2|$,

$$\left\{\langle s_i\rangle_{i\in I} : (\mathbf{s}, st) \leftarrow \mathsf{Scatter}(x_1, n)\right\} \approx \left\{\langle s_i\rangle_{i\in I} : (\mathbf{s}, st) \leftarrow \mathsf{Scatter}(x_2, n)\right\}$$

  where the distributions are over the coins of $\mathsf{Scatter}$. If $t = n$, we sometimes say that f is *fully* hiding. If the distributions are identically distributed we say that f is *perfectly* hiding, and if the distributions are computationally indistinguishable we say f is *computationally* hiding.

If $g = f$, then RR is a *randomized self reduction* (RSR). Furthermore, if there exists a pair of algorithms $\mathsf{RSR} = (\mathsf{Scatter}, \mathsf{Recon})$, such that for every function f in some class $\mathcal{C}$, RSR is a random self reduction for f, then we say that RSR is a *universal* random self reduction over $\mathcal{C}$. All of our constructions are universal.

5.1 A Perfect Randomized Self Reduction for Univariate Polynomials

In this section, we present a *fully*-hiding randomized reduction for univariate polynomials. As far as we know, the best hiding threshold previously achieved by any RR for univariate polynomials of degree q is $t \leq (n-1)/q$ which is achieved by the construction of Beaver, Feigenbaum, Killian and Rogaway [12, 13]. Like the construction presented in [12, 13], our randomized reduction is *universal* and *self-reducing.*

Let $\mathbf{Q}$ be a degree q univariate polynomial over a finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq 2q+1$ and $|\mathbb{F}|^2 - 1 \equiv 1 \pmod 2$, and let $\delta[\mathbb{F}^n] \overset{def}{=} \{\mathbf{v} \in \mathbb{F}^n : v_i \neq v_j \text{ for all } i,j \in [n]\}$. Consider the random self reduction $\mathsf{Poly}^1_\mathsf{q} = (\mathsf{Scatter}_q, \mathsf{Recon}_q)$ defined as follows:

- $\mathsf{Scatter}_q(x)$: let $n = 2q+1$ and sample a vector $\boldsymbol{\alpha}$ uniformly at random from $\delta[\mathbb{F}^n]$. For all $i \in [n]$, compute $s_i := \mathbf{D}_2(\alpha_i, -x/2) = \alpha_i^2 + x$. Output $(s_1, \ldots, s_n)$ and $st = \boldsymbol{\alpha}$.
- $\mathsf{Recon}_q(st, y_1, \ldots, y_n)$: output $y = \sum_{i=1}^n y_i \cdot \mathcal{L}_{\alpha_i}(0)$.

Theorem 5.1. $\mathsf{Poly}^1_\mathsf{q}$ *is a perfect and fully-hiding randomized self reduction.*

Proof. Towards showing correctness, let $\widehat{\mathbf{Q}}(\alpha) \overset{def}{=} \mathbf{Q}\big(\mathbf{D}_2(\alpha, -x/2)\big)$ (for some $x \in \mathbb{F}$) and note that $\widehat{\mathbf{Q}}(0) = \mathbf{Q}(x)$. We therefore have:

$$y = \sum_{i=1}^{2q+1} y_i \cdot \mathcal{L}_{\alpha_i}(0) = \sum_{i=1}^{2q+1} \mathbf{Q}\big(\mathbf{D}_2(\alpha_i, -x/2)\big) \cdot \mathcal{L}_{\alpha_i}(0) = \sum_{i=1}^{2q+1} \widehat{\mathbf{Q}}(\alpha_i) \cdot \mathcal{L}_{\alpha_i}(0) = \widehat{\mathbf{Q}}(0) = \mathbf{Q}(x),$$

since $\deg(\widehat{\mathbf{Q}}) = 2q$. We now consider perfect hiding. Let $n = 2q+1$ and note that for fixed $q \in \mathbb{N}$ and $x \in \mathbb{F}$, $\mathsf{Scatter}$ evaluates the vector-valued function $f_{x,q} : \delta[\mathbb{F}^n] \to \delta[\mathbb{F}^n]$ defined as

$$f_{x,q}(\boldsymbol{\alpha}) = \Big(\mathbf{D}_2\big(\alpha_1, -x/2\big), ..., \mathbf{D}_2\big(\alpha_n, -x/2\big)\Big),$$

for a random $\boldsymbol{\alpha}$. Note that $f_{x,q}$ is a permutation over $\delta[\mathbb{F}^n]$ since $\mathbf{D}_2(\alpha, \beta)$ is a permutation over $\mathbb{F}$ for any β (this follows from the fact that $|\mathbb{F}|^2 - 1 \equiv 1 \pmod 2$). Let $\mathcal{U}$ be the uniform distribution over $\delta[\mathbb{F}^n]$. In the following, for visual clarity we drop the subscript q and denote $f_{x,q}$ by f_x. For all x_1 and x_2 in $\mathbb{F}$,

$$\begin{aligned}
\mathsf{SD}\big(f_{x_1}(\mathcal{U}), f_{x_2}(\mathcal{U})\big) &= \max_{S \subset \delta[\mathbb{F}^n]} \left|\Pr[\, f_{x_1}(\mathcal{U}) \in S\,] - \Pr[\, f_{x_2}(\mathcal{U}) \in S\,]\right| \\
&= \max_{S \subset \delta[\mathbb{F}^n]} \left|\Pr\left[\,\mathcal{U} \in f_{x_1}^{-1}(S)\,\right] - \Pr\left[\,\mathcal{U} \in f_{x_2}^{-1}(S)\,\right]\right| \\
&\leq \max_{\substack{V, V' \subset \delta[\mathbb{F}^n] \\ |V| = |V'|}} \left|\Pr\left[\,\mathcal{U} \in V\,\right] - \Pr\left[\,\mathcal{U} \in V'\,\right]\right| \\
&= 0
\end{aligned}$$

where the last equality follows from the fact that f_{x_1} and f_{x_2} are permutations over $\delta[\mathbb{F}^n]$.

□

5.2 A Computational Randomized Self Reduction for Multivariate Polynomials

We now present a fully-hiding RSR for multi-variate polynomials. The best known hiding threshold previously achieved is from a construction of [12, 13] which achieves $t \leq n \cdot c\log(m)/m$ for c and m greater than 1. Our construction is universal and self-reducing.

Let $\mathbf{Q}$ be a m-variate degree q polynomial over a finite field $\mathbb{F}$ such that $|\mathbb{F}| \geq n+1$, for $n \in \mathbb{N}$. Consider the randomized self reduction $\mathsf{Poly}_\mathsf{q}^\mathsf{m} = (\mathsf{Scatter}_q, \mathsf{Recon}_q)$ defined as follows:

- $\mathsf{Scatter}_q(\mathbf{x})$: let $n = 2q+1$ and sample m univariate polynomials $(p_1, \ldots, p_m)$ of degree 2 such that $p_i(0) = x_i$ for all $i \in [m]$. Let $N = \omega(n \cdot (n/q)^m)$ and $\boldsymbol{\alpha} \overset{\$}{\leftarrow} \delta[\mathbb{F}^n]$. For all $j \in [n]$, set $\mathbf{z}_j := \big(p_1(\alpha_j), \ldots, p_m(\alpha_j)\big)$ and for all $j \in [n+N]$ set $\mathbf{z}_j \overset{\$}{\leftarrow} \mathbb{F}^m$. Let $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_{n+N})$ be the sequence that results from permuting the elements of $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_{n+N})$ at random and let Γ be the locations in $\mathbf{S}$ of the elements in $\mathbf{Z}$ that were chosen at random in $\mathbb{F}^m$. Output $\mathbf{S}$ and $st = (\boldsymbol{\alpha}, \Gamma)$.
- $\mathsf{Recon}_{m,q}(st, y_1, \ldots, y_{n+N})$: parse st as $(\boldsymbol{\alpha}, \Gamma)$ and output $y = \sum_{i \notin \Gamma} y_i \cdot \mathcal{L}_{\alpha_i}(0)$.

The security of our randomized reduction is based on the multi-dimensional noisy curve reconstruction assumption from Ishai, Kushilevitz, Ostrovsky and Sahai [36], which extends the polynomial reconstruction (PR) assumption from Naor and Pinkas [45].

Assumption 5.2 (Multi-dimensional noisy curve reconstruction [36, 45])**.** *The multi-dimensional noisy curve reconstruction (CR) assumption is defined in terms of the following experiment where* $\mathbf{x}$ *is a* m*-dimensional vector over a finite field* $\mathbb{F}$*,* $d > 1$*, and* $t = t(k)$ *and* $z = z(k)$ *are functions of* k*:*

CurveRec$(k, \mathbf{x}, d, n, N, m)$*: sample a vector* $\boldsymbol{\alpha} \overset{\$}{\leftarrow} \mathbb{F}^n$ *and a random subset of* N *indices* Γ *chosen from* $[n+N]$*. Choose* m *random univariate polynomials* $(p_1, \ldots, p_m)$ *such that each* p_i *is of degree at most* d *and that* $p_i(0) = x_i$*. For all* $j \in [n]$*, set* $\mathbf{z}_j = (p_1(\alpha_j), \ldots, p_m(\alpha_j))$ *and for all* $j \in [n+N]$ *set* $\mathbf{z}_j \overset{\$}{\leftarrow} \mathbb{F}^m$*. Let* $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_{n+N})$ *be the sequence that results from permuting the elements of* $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_{n+N})$ *uniformly at random. The output of the experiment is* $(\mathbf{s}_1, \ldots, \mathbf{s}_{n+N})$*.*

We say that the CR assumption holds over $\mathbb{F}$ *with parameters* (d, n, N, m) *if for all* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *in* $\mathbb{F}^m$*,*

$$\Big\{\mathbf{CurveRec}(k, \mathbf{x}_1, d, n, N, m)\Big\} \overset{c}{\approx} \Big\{\mathbf{CurveRec}(k, \mathbf{x}_2, d, n, N, m)\Big\}$$

We note that the CR assumption is believed to hold when N *is* $\omega(n \cdot (n/d)^m)$ *and* $|\mathbb{F}| = N$ *[36].*

Remark. Setting n and N to be polynomial in k, the CR assumption is believed to hold as long as $m = \mathsf{polylog}(k)$. We note, however, that the parameters provided in [36] and used in this work are for the *stronger* "augmented CR" assumption which outputs, in addition to the vectors $(\mathbf{s}_1, \ldots, \mathbf{s}_{n+N})$, the evaluation points $(\alpha_1, \ldots, \alpha_n)$ together with N random values. It is therefore plausible that the CR assumption could hold for a wider range of parameters and, in particular, for $m = \mathsf{poly}(k)$.

In the following theorem, we show that $\mathsf{Poly}_\mathsf{q}^\mathsf{m}$ is a fully-hiding and universal RSR for the class of multi-variate polynomials with a poly-logarithmic number of variables.

Theorem 5.3. $\mathsf{Poly}_\mathsf{q}^\mathsf{m}$ *is a computational and fully-hiding random self reduction.*

Proof. Since the fully-hiding property follows directly from the CR assumption we only show correctness. Let $\widehat{\mathbf{Q}}(\alpha) \overset{def}{=} \mathbf{Q}\big(p_1(\alpha), \ldots, p_m(\alpha)\big)$ and note that $\widehat{\mathbf{Q}}(0) = \mathbf{Q}(x_1, \ldots, x_m)$. We therefore have

$$y = \sum_{i \notin \Gamma} y_i \cdot \mathcal{L}_{\alpha_i}(0) = \sum_{i \notin \Gamma} \mathbf{Q}\big(p_1(\alpha_i), \ldots, p_m(\alpha_i)\big) \cdot \mathcal{L}_{\alpha_i}(0) = \sum_{i \notin \Gamma} \widehat{\mathbf{Q}}(\alpha_i) \cdot \mathcal{L}_{\alpha_i}(0) = \widehat{\mathbf{Q}}(0) = \mathbf{Q}(x_1, \ldots, x_m),$$

since $\deg(\widehat{\mathbf{Q}}) = 2q$ and $|[n+N] \setminus \Gamma| = 2q + 1$. □

6 MR-Parallel HE from Randomized Reductions

We now show how to construct a MR-parallel HE scheme from any $\mathcal{F}$-homomorphic encryption scheme and any fully-hiding RR between functions f and g whose reconstruction algorithm is in $\mathcal{F}$. At a high-level, the construction works as follows.

The RR's scatter algorithm is applied to each element x_i of the input $\mathbf{x}$. This results in a sequence $\mathbf{s}_i$ and a state st_i. The latter is encrypted using the $\mathcal{F}$-homomorphic encryption scheme and each mapper receives a pair composed of a label $\ell = i$ and a value v of the form $(s_i[j], e_i)$ for some $i \in [\#\mathbf{x}]$ and $j \in [n]$ and where e_i is a $\mathcal{F}$-homomorphic encryption of st_i. The mapper evaluates g on $s_i[j]$ and returns an intermediate pair with label $\lambda = i$ and value $\gamma = \big(g(s_i[j]), e_i\big)$. After the shuffle operation, each reducer receives a pair composed of a label i and a partition

$$P = \Big((y_{i,j}, e_i), \ldots, (y_{i,n}, e_i) \Big),$$

where $y_{i,j} = g(s_i[j])$ for $j \in [n]$. Since Recon is in $\mathcal{F}$, the reducer can evaluate $\mathsf{Recon}(e_i, y_{i,1}, \ldots, y_{i,n})$ homomorphically which results in an encryption of $f(x_i)$.

Note that this approach yields a function-private MR-parallel HE scheme so it can be used in the context of private function evaluation. To be useful for server-aided computation, however, the running time of the reduction's scatter algorithm and of the HE scheme's decryption algorithm must be less than the fastest known algorithm for computing f. If the reduction is $\mathcal{C}$-universal, however, the client can evaluate *multiple* functions on the same encrypted input if we include a description of the function in the input value. We describe this variant in Figure 1.

Theorem 6.1. *If* HE *is CPA-secure and if* RR *is fully-hiding, then* PHE *as described in Figure 1 is secure against single-message chosen-plaintext attacks.*

Proof sketch: Consider the simulator $\mathcal{S}$ that simulates ciphertexts in an $\mathbf{Ideal}(k)$ experiment as follows. Given $\#\mathbf{x}$ it generates $(pk', sk') \leftarrow \mathsf{Gen}(1^k)$ and, for all $i \in [\#\mathbf{x}]$, it computes $(\mathbf{s}'_i, st'_i) \leftarrow \mathsf{Scatter}(0)$ and $e'_i \leftarrow \mathsf{HE.Enc}_{pk'}(st'_i)$. It outputs $\mathbf{c}' = (pk', \mathbf{s}'_1, \ldots, \mathbf{s}'_{\#\mathbf{x}}, e'_1, \ldots, e'_{\#\mathbf{x}})$. The fully-hiding property of RR guarantees that the $\mathbf{s}'_i$'s are indistinguishable from the $\mathbf{s}_i$'s generated in a $\mathbf{Real}(k)$ experiment. Similarly, the CPA-security of HE guarantees that the e'_i's are indistinguishable from the e_i's generated in a $\mathbf{Real}(k)$ experiment. □

7 An Efficient Delegated MR-Parallel HE Scheme for $\mathbf{NC}^0_{1,\mathsf{eq}}$

It is easy to see that instantiating the RR and the HE scheme in our general construction with our universal and fully-hiding RSR for univariate polynomials (from section 5.1) and a FHE scheme yields a

Let $\mathsf{HE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a public-key $\mathcal{F}$-homomorphic encryption scheme and let $\mathsf{RR} = (\mathsf{Scatter}, \mathsf{Recon})$ be a $\mathcal{C}$-universal (t, n)-local randomized reduction from f to g such that $\mathsf{Recon} \in \mathcal{F}$. Consider the multi-use MR-parallel $\mathcal{C}$-homomorphic encryption scheme $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, where $\mathsf{PHE.Eval} = (\mathsf{Parse}, \mathsf{Map}, \mathsf{Part}, \mathsf{Red}, \mathsf{Merge})$, defined as follows:

- $\mathsf{Gen}(1^k)$: compute $(pk, sk) \leftarrow \mathsf{HE.Gen}(1^k)$. Output $K = (sk, pk)$.
- $\mathsf{Enc}(K, \mathbf{x})$: for all $i \in [\#\mathbf{x}]$, compute $(\mathbf{s}_i, st_i) \leftarrow \mathsf{Scatter}(x_i)$ and $e_i \leftarrow \mathsf{HE.Enc}_{pk}(st_i)$. Output $\mathbf{c} = (pk, \mathbf{s}_1, \ldots, \mathbf{s}_{\#\mathbf{x}}, e_1, \ldots, e_{\#\mathbf{x}})$.
- $\mathsf{Parse}(f, \mathbf{c})$: for all $i \in [\#\mathbf{x}]$ and $j \in [n]$, set $\ell_{i,j} := i$ and $v_{i,j} := (f, pk, \mathbf{s}_i[j], e_i)$. Output $(\ell_{i,j}, v_{i,j})_{i,j}$.
- $\mathsf{Map}(\ell, v)$: parse v as (f, s, e) and compute $a \leftarrow \mathsf{HE.Enc}_{pk}(g(s))$. Output $\lambda := \ell$ and $\gamma := (a, e)$.
- $\mathsf{Red}(\lambda, P)$: parse P as $(a_r, e_r)_r$ and compute $z \leftarrow \mathsf{HE.Eval}(\mathsf{Recon}, e_r, (a_r)_r)$. Output (λ, z).
- $\mathsf{Merge}\big((\lambda_t, z_t)_t\big)$: output $\mathbf{c}' := (z_t)_t$.
- $\mathsf{Dec}(K, \mathbf{c}')$: for all $i \in [\#\mathbf{c}']$, compute $y_i := \mathsf{HE.Dec}_{sk}(z_i)$. Output $\mathbf{y} = (y_1, \ldots, y_{\#\mathbf{c}'})$.

Figure 1: MR-parallel HE from RR and HE

multi-use MR-parallel HE scheme for functions in $\mathrm{NC}^0_{1,\mathsf{eq}}$ since all functions $f \in \mathrm{NC}^0_{1,\mathsf{eq}}$ can be computed by evaluating a single univariate polynomial. In addition, the resulting construction can be made delegated by encrypting the coefficients of the polynomial using the FHE scheme and having the mappers perform their computations homomorphically.

Current FHE constructions, however, are not yet practical enough for our purposes so we present a direct construction based only on 2DNF-HE. The direct construction also has the advantage that the input values sent to the mappers are smaller than what would result from our general construction. We describe our scheme in detail in Figure 2 but, at a high level, it works as follows. First, recall that $\mathsf{Poly}^1_\mathsf{q}$ scatters an input x by using it to choose a polynomial $\mathbf{D}$ from a family of permutation polynomials of degree 2. This polynomial is then evaluated at $n = 2q+1$ distinct locations $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ chosen uniformly at random. This results in a sequence $\mathbf{s} = (s_1, \ldots, s_n)$ which, together with $\boldsymbol{\alpha}$, are the output of the scatter operation. Reconstruction consists recovering $\mathbf{Q}(x)$ from $\mathbf{y} = \big(\mathbf{Q}(s_1), \ldots, \mathbf{Q}(s_n)\big)$ and $\boldsymbol{\alpha}$ using interpolation. The main difficulty in using the general approach is that it is unclear how to evaluate the reconstruction algorithm (i.e., interpolation of $\mathbf{Q}(x)$ from $\mathbf{y}$ and $\boldsymbol{\alpha}$) homomorphically without making use of FHE.

Our approach, therefore, will be to have the client perform some additional work in order to make the reconstruction algorithm simpler and computable using a weaker HE scheme. In particular, the client will itself compute the Lagrangians needed for the interpolation and include them (encrypted) as part of the ciphertext. Notice that given $\mathbf{y}$ and a sequence of encrypted Lagrangians $\big(\mathsf{Enc}_K(\mathcal{L}_{\alpha_1}(0)), \ldots, \mathsf{Enc}_K(\mathcal{L}_{\alpha_n}(0))\big)$, the value $\mathbf{Q}(x)$ can be interpolated homomorphically using only scalar multiplication and addition since

$$\mathbf{Q}(x) = \sum_i y_i \cdot \mathcal{L}_{\alpha_i}(0).$$

To make the scheme delegated, however, we would also like to encrypt the coefficients of $\mathbf{Q}$ in such a way that $y_i = \mathbf{Q}(s_i) = \sum_{j=0}^n a_j \cdot s_i^j$ can be evaluated homomorphically. These two constraints essentially require that we be able to perform one multiplication (between an encrypted y_i and an encrypted Lagrangians) and multiple additions homomorphically which we can do relatively efficiently using a 2DNF-HE scheme.

In the following Theorem we prove the security of our construction.

Let $\mathsf{DNF} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a private-key 2DNF-homomorphic encryption scheme and consider the delegated MR-parallel $\mathrm{NC}^0_{1,\mathsf{eq}}$-homomorphic encryption scheme $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, where $\mathsf{PHE.Eval} = (\mathsf{Token}, \mathsf{Parse}, \mathsf{Map}, \mathsf{Shuffle}, \mathsf{Red}, \mathsf{Merge})$, defined as follows:

- $\mathsf{Gen}(1^k)$: compute and output $K \leftarrow \mathsf{DNF.Gen}(1^k)$.
- $\mathsf{Enc}(K, \mathbf{x})$: for all $i \in [\#\mathbf{x}]$ sample $\boldsymbol{\alpha}_i \xleftarrow{\$} \delta[\mathbb{F}^n]$ and compute

$$\mathbf{s}_i := \Big(\mathbf{D}_2\big(\alpha_1, -x_i/2\big), \ldots, \mathbf{D}_2\big(\alpha_n, -x_i/2\big)\Big)$$

  and

$$\mathbf{e}_i := \Big(\mathsf{Enc}_K\big(\mathcal{L}_{\boldsymbol{\alpha}_i[1]}(0)\big), \ldots, \mathsf{Enc}_K\big(\mathcal{L}_{\boldsymbol{\alpha}_i[n]}(0)\big)\Big),$$

  where Enc refers to the encryption scheme of DNF. Output $\mathbf{c} = \big((\mathbf{s}_i[j], \mathbf{e}_i[j])\big)_{i,j}$.
- $\mathsf{Token}(K, f)$: parse f as $(\mathbf{Q}, \ldots, \mathbf{Q})$ and let $(a_0, \ldots, a_q)$ be the coefficients of $\mathbf{Q}$. For $0 \le i \le q$, compute $\tau_i \leftarrow \mathsf{DNF.Enc}_K(a_i)$. Output $\boldsymbol{\tau} := (\tau_0, \ldots, \tau_q)$.
- $\mathsf{Parse}(\boldsymbol{\tau}, \mathbf{c})$: For $i \in [\#\mathbf{x}]$ and $j \in [n]$, set $\ell_{i,j} := i$ and $v_{i,j} := (\boldsymbol{\tau}, \mathbf{s}_i[j], e_i[j])$. Output $(\ell_{i,j}, v_{i,j})_{i,j}$.
- $\mathsf{Map}(\ell, v)$: parse v as $(\boldsymbol{\tau}, s, e)$ and compute

$$h := \mathsf{Eval}(+, \tau_0, \tau_1 \cdot s, \ldots, \tau_q \cdot s^q) = \mathsf{Enc}_K\left(\sum_{i=0}^{q} a_i s^i\right) = \mathsf{Enc}_K\big(\mathbf{Q}(s)\big),$$

  and

$$\gamma := \mathsf{Eval}(\times, h, e) = \mathsf{Enc}_K\Big(\mathbf{Q}(s) \cdot \mathcal{L}_{\alpha_j}(0)\Big),$$

  where Eval and Enc refer to the evaluation and encryption algorithms of DNF and where j is some value in $[n]$. Output $\lambda := \ell$ and γ.
- $\mathsf{Red}(\lambda, P)$: parse P as $(\gamma_1, \ldots, \gamma_n)$ and output

$$z := \mathsf{Eval}(+, \gamma_1, \ldots, \gamma_n) = \mathsf{Enc}_K\left(\sum_{i=1}^{n} \mathbf{Q}(s_i) \cdot \mathcal{L}_{\alpha_i}(0)\right) = \mathsf{Enc}_K\big(\mathbf{Q}(x_i)\big),$$

  for some $i \in [n]$ and where Eval and Enc refer to the evaluation and encryption algorithms of DNF.
- $\mathsf{Merge}\big((\lambda_t, z_t)_t\big)$: output $\mathbf{c}' := (\lambda_t, z_t)_t$.
- $\mathsf{Dec}(K, \mathbf{c}')$: for all $i \in [\#\mathbf{c}']$, compute $y_i := \mathsf{HE.Dec}_K(z_i)$. Output $\mathbf{y} = (y_1, \ldots, y_{\#\mathbf{c}'})$.

Figure 2: A delegated MR-parallel $\mathrm{NC}^0_{1,\mathsf{eq}}$-homomorphic encryption.

Theorem 7.1. *If* DNF *is CPA-secure, then* PHE *as described in Figure 2 is secure against single-input chosen-plaintext and function attacks.*

Proof sketch: Consider the simulator $\mathcal{S}$ that simulates ciphertexts and tokens in an $\mathbf{Ideal}^\star(k)$ experiment as follows. Given $\#\mathbf{x}$, it generates a key $K' \leftarrow \mathsf{DNF.Gen}(1^k)$ and, for all $i \in [\#\mathbf{x}]$ and $j \in [n]$, computes

$(\mathbf{s}'_i, \boldsymbol{\alpha}'_i) \leftarrow \mathsf{Poly}^1_\mathsf{q}.\mathsf{Scatter}(0)$ and

$$\mathbf{e}'_i := \Big(\mathsf{Enc}_K\big(\mathcal{L}_{\boldsymbol{\alpha}'_i[1]}(0)\big), \ldots, \mathsf{Enc}_{K'}\big(\mathcal{L}_{\boldsymbol{\alpha}'_i[n]}(0)\big)\Big).$$

It then returns $\mathbf{c}' = (\mathbf{s}'_i[j], \mathbf{e}'_i[j])_{i,j}$. Given q, it returns $\boldsymbol{\tau}' = (\tau'_1, \ldots, \tau'_q)$, where $\tau'_i = \mathsf{Enc}_{K'}(0)$.

The fully-hiding property of $\mathsf{Poly}^1_\mathsf{q}$ guarantees that the $s'_{i,j}$'s are indistinguishable from the $s_{i,j}$'s generated in a **Real*** experiment. Similarly, the CPA-security of DNF guarantees that the $e'_{i,j}$'s are indistinguishable from the $e_{i,j}$'s in a **Real**$^\star(k)$ experiment. Finally, the CPA-security of DNF also guarantees that each $\boldsymbol{\tau}'$ is indistinguishable from the $\boldsymbol{\tau}$ generated in a **Real**$^\star(k)$ experiment. □

Efficiency. The most efficient approach for evaluating a polynomial on multiple points using the Fast Fourier Transform (FFT) takes time $O(q \log q)$ for the evaluation of a degree q polynomial on q points. Thus, the costs to evaluate a univariate polynomial $\mathbf{Q}$ of degree q over a sequence $\mathbf{x}$ using our construction are $O(\#\mathbf{x} \cdot \log q)$ for encryption, $O(q)$ for token generation and $O(\#\mathbf{x})$ for decryption.

Therefore, in the setting of server-aided computation, our scheme can reduce the client's work from $O(w \cdot \#\mathbf{x} \cdot \log q)$ to $O(\#\mathbf{x} \cdot \log q + w \cdot (q + \#\mathbf{x}))$ when evaluating $f = (\mathbf{Q}_1, \ldots, \mathbf{Q}_w)$ in $\mathrm{NC}^0_{1,\mathsf{eq}}$ and $\#\mathbf{x} \gg q$.

8 A MR-Parallel HE Scheme for NC^0_m

Similarly to the case of $\mathrm{NC}^0_{1,\mathsf{eq}}$ functions, our general construction yields a multi-use MR-parallel HE scheme for functions in NC^0_m if we instantiate its RR and HE scheme with a fully-hiding RR for multivariate polynomials (as our construction in section 5.2) and a FHE scheme.

To avoid the use of FHE, we present a direct construction that only makes use of additively HE. The high-level approach is similar to our direct construction for $\mathrm{NC}^0_{1,\mathsf{eq}}$ functions: we make the client compute the Lagrangians in order to simplify the computation that has to be performed homomorphically by the reducers. Our construction is described in detail in Figure 3.

Theorem 8.1. *If* AHE *is CPA-secure and if the multi-dimensional noisy curve reconstruction assumption holds, then* PHE *as described in Figure 3 is secure against single-message chosen-plaintext attacks.*

Proof sketch: Consider the simulator $\mathcal{S}$ that simulates ciphertexts in an **Ideal**(k) experiment as follows. Given $\#\mathbf{x}$ and q, it starts by generating a key $K' \leftarrow \mathsf{AHE.Gen}(1^k)$. Then, for all $i \in [\#\mathbf{x}]$ it (1) generates $\mathbf{s}'_i$ by running a $\mathbf{CurveRec}(k, \mathbf{0}, 2, n, N, m)$ experiment (where $\mathbf{0} \in \mathbb{F}^m$); and (2) computes $\mathbf{e}'_i := \big(\mathsf{Enc}_{K'}(0), \ldots, \mathsf{Enc}_{K'}(n+N)\big)$, where Enc refers to the encryption algorithm of AHE. Finally, $\mathcal{S}$ outputs the ciphertext $\mathbf{c}' := (\mathbf{s}'_i, \mathbf{e}'_i)_i$.

The CR assumption and the CPA-security of AHE guarantee that the $\mathbf{s}'_i$'s and $\mathbf{e}'_i$'s generated by $\mathcal{S}$ are indistinguishable from the $\mathbf{s}_i$'s and $\mathbf{e}_i$'s in a **Real**(k) experiment. □

Efficiency. The costs to evaluate a function $f \in \mathrm{NC}^0_m$ of degree q and such that $f = (\mathbf{Q}_1, \ldots, \mathbf{Q}_w)$ using our construction are $O(\#\mathbf{x} \cdot m \cdot q)$ for encryption and $O(w)$ for decryption. Therefore, using our scheme, the client can reduce its work from $O(\#\mathbf{x} \cdot q^m)$ (using, e.g., the multivariate Horner rule) to $O(\#\mathbf{x} \cdot m \cdot q + w)$.

Let $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a private-key additively homomorphic encryption scheme and consider the MR-parallel NC^0_m-homomorphic encryption scheme $\mathsf{PHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, where $\mathsf{PHE.Eval} = (\mathsf{Parse}, \mathsf{Map}, \mathsf{Shuffle}, \mathsf{Red}, \mathsf{Merge}, \mathsf{Dec})$, defined as follows:

- $\mathsf{Gen}(1^k)$: output $K \leftarrow \mathsf{AHE.Gen}(1^k)$.
- $\mathsf{Enc}(K, \mathbf{x})$:

    1. let $n = 2q + 1$ and $N = \omega(n \cdot (n/q)^m)$
    2. choose a random permutation π over $[n + N]$
    3. sample a vector $\boldsymbol{\alpha} \overset{\$}{\leftarrow} \delta[\mathbb{F}^n]$
    4. for all $i \in [\#\mathbf{x}]$:
        (a) sample a univariate polynomial p_i at random such that $p_i(0) = x_i$
        (b) for all $j \in [n]$,
            i. compute $z_j := p_i(\alpha_j)$ and $z'_j \leftarrow \mathsf{AHE.Enc}_K(\mathcal{L}_{\alpha_j}(0))$
        (c) for all $j \in [n + 1, n + N]$,
            i. sample $z_j \overset{\$}{\leftarrow} \mathbb{F}$ and compute $z'_j \leftarrow \mathsf{AHE.Enc}_K(0)$.
        (d) let $\mathbf{s}_i := \pi(\mathbf{z})$ and $\mathbf{e}_i := \pi(\mathbf{z}')$
        (e) let Γ be the locations in $\mathbf{s}_i$ of the elements in $\mathbf{z}$ that were chosen at random
    5. Output $\mathbf{c} := (\mathbf{s}_i, \mathbf{e}_i)_i$

- $\mathsf{Parse}(f, c)$: parse f as $(\mathbf{Q}_1, \ldots, \mathbf{Q}_w)$ and denote the locations on which $\mathbf{Q}$ depends by $\mathsf{in}(\mathbf{Q})$. For all $i \in [w]$ and all $j \in [n + N]$, set $\ell_{i,j} = \mathbf{Q}_i$ and $v_{i,j} = \big((\mathbf{s}_\mu[j])_{\mu \in \mathsf{in}(\mathbf{Q}_i)}, \mathbf{e}_i[j]\big)$. Output $(\ell_{i,j}, v_{i,j})_{i,j}$.
- $\mathsf{Map}(\ell, v)$: parse ℓ as $\mathbf{Q}$ and v as $(\mathbf{s}, e)$ and set $\lambda := \ell = \mathbf{Q}$ and $\gamma := \mathbf{Q}(\mathbf{s}) \cdot e$. Output (λ, γ).
- $\mathsf{Red}(\lambda, P)$: parse λ as $\mathbf{Q}$ and P as $(\gamma_1, \ldots, \gamma_{n+N})$ and output

$$z := \mathsf{Eval}(+, \gamma_1, \ldots, \gamma_{n+N}) = \mathsf{Enc}_K\left(\sum_{j=1}^{n+N} \mathbf{Q}\big(\mathbf{s}_{i_1}[j], \ldots, \mathbf{s}_{i_m}[j]\big) \cdot \zeta_{i,j}\right) = \mathsf{Enc}_K\big(\mathbf{Q}(\mathbf{x})\big),$$

  where $(i_1, \ldots, i_m)$ are the indices of the elements on which $\mathbf{Q}$ depends and where $\zeta_{i,j} = \mathcal{L}_{\alpha_j}\big(0\big)$ if $j \notin \Gamma$ and $\zeta_{i,j} = 0$ if $j \in \Gamma$; and Eval and Enc refer to the evaluation and encryption algorithms of AHE.
- $\mathsf{Merge}((\lambda_t, z_t)_t)$: output $\mathbf{c}' = (\lambda_t, z_t)_t$.
- $\mathsf{Dec}(K, \mathbf{c}')$: for $t \in [w]$, compute $y_t := \mathsf{AHE.Dec}_K(z_t)$. Output $y := (y_1, \ldots, y_w)$.

Figure 3: A MR-parallel NC^0_m-homomorphic encryption scheme for $m = \mathsf{polylog}(k)$.

9 Applications of Our Constructions

In this section we discuss applications of our MR-parallel HE schemes. These applications do not depend on any particular feature of our construction but can be achieved using any MR-parallel HE scheme for $\mathrm{NC}^0_{1,\mathsf{eq}}$ and NC^0_m. We show how to perform various queries over a n-element database $\mathbf{x} \in \mathbb{F}^n$.

Simple database queries. The simplest query we can support is for set membership. More precisely, if $S \subseteq \mathbb{F}$, the query $\mathsf{Set}(\mathbf{x}, S)$ returns a n-bit string such that the ith bit indicates whether $x_i \in S$. Note that the set S could encode a numerical range or a list of keywords. Our approach here is similar to that of Kissner and Song for the purpose of secure two-party set intersection [41]. Using our $\mathrm{NC}^0_{1,\mathsf{eq}}$-HE scheme, it suffices to evaluate the function $f = (\mathbf{Q}_1, \ldots, \mathbf{Q}_n)$ on $\mathbf{x}$, where $\mathbf{Q}_1 = \ldots = \mathbf{Q}_n$ is a univariate polynomial whose roots are the values in S. Correctness follows from the fact that if $x_i \in S$ then $\mathbf{Q}_i(x_i) = 0$ and if $x_i \notin S$ then $\mathbf{Q}_i(x_i) \neq 0$.

A slightly more complex query is $\mathsf{OR}(\mathbf{x}, i_1, \ldots, i_m, w_1, \ldots x_m)$ which outputs

$$\Big(x_{i_1} = w_1 \bigwedge \cdots \bigwedge x_{i_m} = w_m\Big).$$

The OR query can be computed using our NC^0_m-HE scheme by evaluating the function $\mathbf{Q}(x_{i_1}, \ldots, x_{i_m}) = (w_1 - x_{i_1}) \times \cdots \times (w_m - x_{i_m})$. Correctness follows from the fact that $\mathbf{Q}(x_{i_1}, \ldots, x_{i_m}) = 0$ if $x_{i_j} = w_j$ for some j and $\mathbf{Q}(x_{i_1}, \ldots, x_{i_m}) \neq 0$ otherwise.

Keyword search. A more complex query that can also be performed using our NC^0_m construction with $m = 2$ is keyword search. Here, let $\mathbf{x} = \big((w_1, v_1), \ldots, (w_n, v_n)\big)$ be a dataset where w_i and v_i are in $\mathbb{F}$. The query $\mathsf{KS}(\mathbf{x}, w)$ outputs a sequence $(z_1, \ldots, z_n)$ such that $z_i = v_i$ if $w_i = w$ and such that $z_i = 0$ if $w_i \neq w$. The KS query can be handled using the approach of [26] which for every pair (w_i, v_i) computes the polynomial $\mathbf{Q}_{w,r}(w_i, v_i) = p_1(v_i) + r \cdot p_2(w_i)$, where $r \overset{\$}{\leftarrow} \mathbb{F}$, $p_1(v)$ returns $v||0^k$ and $p_2(w) = 0$.

Acknowledgements

The authors are grateful to Yuval Ishai and Payman Mohassel for useful comments and suggestions.

References

[1] Amazon elastic mapreduce. See `http://aws.amazon.com/elasticmapreduce`.

[2] Cloudera. See `http://cloudera.com`.

[3] Powered by hadoop. See `http://wiki.apache.org/hadoop/PoweredBy`.

[4] Yahoo! launches world's largest haddoop production application. See `http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html`, 2008.

[5] M. Abadi, J. Feingenbaum, and J. Killian. On hiding information from an oracle. In *ACM Symposium on Theory of Computing (STOC 1987)*, pages 195–203, 1987.

[6] B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. *Journal of Computational Complexity*, 15(2), 2006.

[7] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC0. *SIAM Journal of Computation*, 36(4):845–888, December 2006.

[8] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. In *Advances in Cryptology - CRYPTO '07*, Lecture Notes in Computer Science, pages 92–110. Springer-Verlag, 2007.

[9] B. Applebaum, Y. Ishai, and E. Kushilevitz. On pseudorandom generators with linear stretch in nc0. *Journal of Computational Complexity*, 17(1):38–69, 2008.

[10] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages and Programming (ICALP '10)*, pages 152–163, 2010.

[11] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Symposium on Theoretical aspects of Computer Science (STACS '90)*, pages 37–48. Springer, 1990.

[12] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with low communication overhead. In *Advances in Cryptology - CRYPTO '90*, pages 62–76. Springer-Verlag, 1991.

[13] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *Journal of Cryptology*, 10(1):17–36, 1997.

[14] J. Benaloh. *Verifiable secret-ballot elections.* PhD thesis, Yale University, 1987.

[15] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference (TCC '05)*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–342. Springer, 2005.

[16] A. Borodin. On relating time and space to size and depth. *SIAM Journal of Computation*, 6(4):733–744, 1977.

[17] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *International Colloquium on Automata, Languages and Programming (ICALP '00)*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523. Springer, 2000.

[18] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *Conference on World Wide Web*, pages 231–240, New York, NY, USA, 2010. ACM.

[19] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Conference on Neural Information Processing Systems (NIPS '06)*, pages 281–288. MIT Press, 2006.

[20] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. von Eicken. Logp: a practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, 1996.

[21] I. Damgard and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *International Workshop on Practice and Theory in Public Key Cryptography (PKC '01)*, pages 119–136. Springer-Verlag, 2001.

[22] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Conference on World Wide Web (WWW '07)*, pages 271–280, New York, NY, USA, 2007. ACM.

[23] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[24] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[25] S. Fortune and J. Wyllie. Parallelism in random access machines. In *ACM Symposium on Theory of Computing (STOC '78)*, pages 114–118, New York, NY, USA, 1978. ACM.

[26] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and obliviosu pseudorandom functions. In *Theory of Cryptography Conference (TCC '05)*. Springer, 2005.

[27] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO 1984*, Lecture Notes in Computer Science, pages 10–18. Springer, 1985.

[28] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[29] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[30] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology - CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2010.

[31] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple bgn-type cryptosystem from lwe. In *Advances in Cryptology - EUROCRYPT '10*, pages 506–522. Springer, 2010.

[32] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[33] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*, pages 59–72, New York, NY, USA, 2007. ACM.

[34] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *IEEE Symposium on Foundations of Computer Science (FOCS '00)*, pages 294–304. IEEE Press, 2000.

[35] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *International Colloquium on Automata, Languages and Programming (ICALP '02)*, pages 244–256. Springer, 2002.

[36] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *IEEE Symposium on Foundations of Computer Science (FOCS '06)*. IEEE Computer Society, 2006.

[37] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference (TCC '07)*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[38] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *International Conference on Data Mining (ICDM '09)*, pages 229–238, Washington, DC, USA, 2009. IEEE Computer Society.

[39] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Symposium on Discrete Algorithms (SODA '10)*, pages 938–948. SIAM, 2010.

[40] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. pages 869–941, 1990.

[41] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, 2005.

[42] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool, 2010.

[43] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in mapreduce. In *Workshop on Mining and Learning with Graphs*, pages 78–85, New York, NY, USA, 2010. ACM.

[44] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *ACM Conference on Computer and Communications Security (CCS '98)*, pages 59–66. ACM Press, 1998.

[45] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal of Computation*, 35(5):1254–1281, 2006.

[46] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 1998.

[47] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[48] D. Rao and D. Yarowsky. Ranking and semi-supervised classification on large scale graphs using map-reduce. In *Workshop on Graph-based Methods for Natural Language Processing*, pages 58–65, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

[49] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for nc1. In *IEEE Symposium on Foundations of Computer Science (FOCS '99)*, page 554. IEEE Computer Society, 1999.

[50] N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Conference on Practice and Theory in Public Key Cryptography (PKC '10)*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[51] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[52] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.