Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy

Qian Yu*, Songze Li*, Netanel Raviv†, Seyed Mohammadreza Mousavi Kalan*, Mahdi Soltanolkotabi*, and A. Salman Avestimehr*

* Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA

† Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, USA

Abstract

We consider a scenario involving computations over a massive dataset stored distributedly across multiple workers, which is at the core of distributed learning algorithms. We propose *Lagrange Coded Computing* (LCC), a new framework to *simultaneously* provide (1) *resiliency* against stragglers that may prolong computations; (2) *security* against Byzantine (or malicious) workers that deliberately modify the computation for their benefit; and (3) (information-theoretic) *privacy* of the dataset amidst possible collusion of workers. LCC, which leverages the well-known Lagrange polynomial to create computation redundancy in a novel coded form across workers, can be applied to any computation scenario in which the function of interest is an *arbitrary multivariate polynomial* of the input dataset, hence covering many computations of interest in machine learning. LCC significantly generalizes prior works to go beyond linear computations. It also enables secure and private computing in distributed settings, improving the computation and communication efficiency of the state-of-the-art. Furthermore, we prove the optimality of LCC by showing that it achieves the optimal tradeoff between resiliency, security, and privacy, i.e., in terms of tolerating the maximum number of stragglers and adversaries, and providing data privacy against the maximum number of colluding workers. Finally, we show via experiments on Amazon EC2 that LCC speeds up the conventional uncoded implementation of distributed least-squares linear regression by up to $13.43\times$, and also achieves a $2.36\times$-$12.65\times$ speedup over the state-of-the-art straggler mitigation strategies.

I. Introduction

The massive size of modern datasets necessitates computational tasks to be performed in a distributed fashion, where the data is dispersed among many servers that operate in parallel [1]. As we "scale out" computations across many servers, however, several fundamental challenges arise. Cheap commodity hardware tends to vary greatly in computation time, and it has been demonstrated [2]–[4] that a small fraction of servers, referred to as *stragglers*, can be 5 to 8 times slower than the average, thus creating significant delays in computations. Also, as we distribute computations across many servers, massive amounts data must be moved between them to execute the computational tasks, often over many iterations of a running algorithm, and this creates a substantial bandwidth bottleneck [5]. Distributed computing systems are also much more susceptible to adversarial servers, making security and privacy a major concern [6]–[8].

We consider a general scenario in which the computation is carried out distributively across several workers, and propose *Lagrange Coded Computing* (LCC), a new framework to simultaneously provide

1) *resiliency* against straggler workers that may prolong computations;
2) *security* against Byzantine (or malicious, adversarial) workers, with no computational restriction, that deliberately send erroneous data in order to affect the computation for their benefit; and
3) *(information-theoretic) privacy* of the dataset amidst possible collusion of workers.

LCC can be applied to any computation scenario in which the function of interest is an *arbitrary multivariate polynomial* of the input dataset. This covers many computations of interest in machine learning, such as various gradient and loss-function computations in learning algorithms and tensor algebraic operations (e.g., low-rank tensor approximation). The key idea of LCC is to encode the input dataset using the well-known Lagrange polynomial, in order to create computational redundancy in a novel coded form across the workers. This redundancy can then be exploited to provide resiliency to stragglers, security against malicious servers, and privacy of the dataset.

Specifically, as illustrated in Fig. 1, using a master-worker distributed computing architecture with N workers, the goal is to compute $f(X_i)$ for every X_i in a large dataset $X = (X_1, X_2, \ldots, X_K)$, where f is a given multivariate polynomial with degree $\deg f$. To do so, N coded versions of the input dataset, denoted by $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_N$ are created, and the workers then compute f over the coded data, *as if no coding is taking place*. For a given N and f, we say that the tuple (S, A, T) is *achievable* if there exists an encoding and decoding scheme that can complete the computations in the presence of up to S stragglers, up to A adversarial workers, whilst keeping the dataset private against sets of up to T colluding workers.

Our main result is that by carefully encoding the dataset the proposed LCC achieves (S, A, T) if $(K+T-1)\deg f+S+2A+1 \leq N$. The significance of this result is that by one additional worker (i.e., increasing N by 1) LCC can increase the resiliency to stragglers by 1 or increase the robustness to malicious servers by $1/2$, while maintaining the privacy constraint. Hence, this result essentially extends the well-known optimal scaling of error-correcting codes (i.e., adding one parity can provide

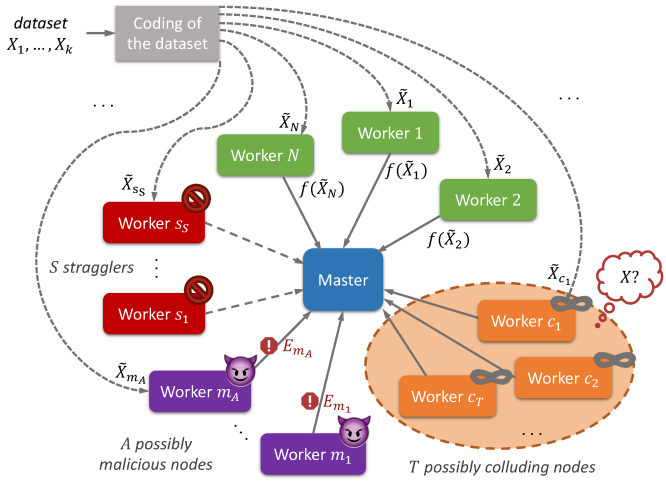

Figure 1. An overview of the problem considered in this paper, where the goal is to evaluate a *not necessarily linear* function f on a given dataset $X = (X_1, X_2, \ldots, X_K)$ using N workers. Each worker applies f on a possibly *coded* version of the inputs (denoted by $\tilde{X}_i$'s). By carefully designing the coding strategy, the master can decode all the required results from a subset of workers, in the presence of *stragglers* (workers $s_1, ..., s_S$) and *Byzantine workers* (workers $m_1, ..., m_A$), while keeping the dataset private to *colluding workers* (workers $c_1, ..., c_T$).

robustness against one erasure or $1/2$ error in optimal maximum distance separable codes) to the distributed secure computing paradigm.

We prove the optimality of LCC by showing that it achieves the optimal tradeoff between resiliency, security, and privacy. In other words, any computing scheme (under certain complexity constrains on the encoding and decoding designs) can achieve (S, A, T) if and only if $(K + T - 1) \deg f + S + 2A + 1 \leq N$.[1] This result further extends the scaling law in coding theory to private computing, showing that any additional worker enables data privacy against $1/\deg f$ additional colluding workers.

Finally, we specialize our general theoretical guarantees for LCC in the context of least-squares linear regression, which is one of the elemental learning tasks, and demonstrate its performance gain by optimally suppressing stragglers. Leveraging the algebraic structure of gradient computations, several strategies have been developed recently to exploit data and gradient coding for straggler mitigation in the training process (see, e.g., [9]–[13]). We implement LCC for regression on Amazon EC2 clusters, and empirically compare its performance with the conventional uncoded approaches, and two state-of-the-art straggler mitigation schemes: gradient coding (GC) [10], [14]–[16] and matrix-vector multiplication (MVM) based approaches [9], [11]. Our experiment results demonstrate that compared with the uncoded scheme, LCC improves the run-time by $6.79\times$-$13.43\times$. Compared with the GC scheme, LCC improves the run-time by $2.36\times$-$4.29\times$. Compared with the MVM scheme, LCC improves the run-time by $1.01\times$-$12.65\times$.

Related works. There has recently been a surge of interest on using coding theoretic approaches to alleviate key bottlenecks (e.g., stragglers, bandwidth, and security) in distributed machine learning applications (e.g., [10], [14], [15], [17]–[25]). As we discuss in more detail in Section III-A, the proposed LCC scheme significantly advances prior works in this area by 1) generalizing coded computing to arbitrary multivariate polynomial computations, which are of particular importance in learning applications; 2) extending the application of coded computing to secure and private computing; 3) reducing the computation/communication load in distributed computing (and distributed learning) by factors that scale with the problem size, without compromising security and privacy guarantees; and 4) enabling $2.36\times$-$12.65\times$ speedup over the state-of-the-art in distributed least-squares linear regression in cloud networks.

Secure *multiparty computing* (MPC) and secure/private Machine Learning (e.g., [26], [27]) are also extensively studied topics that address a problem setting similar to LCC. As we elaborate in Section III-A, compared with conventional methods in this area (e.g., the celebrated BGW scheme for secure/private MPC [26]), LCC achieves substantial reduction in the amount of randomness, storage overhead, and computation complexity.

II. Problem Formulation and Examples

We consider the problem of evaluating a multivariate polynomial $f : \mathbb{V} \to \mathbb{U}$ over a dataset $X = (X_1, \ldots, X_K)$,[2] where $\mathbb{V}$ and $\mathbb{U}$ are vector spaces of dimensions M and L, respectively, over the field $\mathbb{F}$. We assume a distributed computing environment

[1] More accurately, when $N < K \deg f - 1$, we prove that the optimal tradeoff is instead given by $K(S + 2A + \deg f \cdot T + 1) \leq N$, which can be achieved by a variation of the LCC scheme, as described in Appendix D.

[2] We focus on the non-trivial case where $K > 0$ and f is not constant.

with a master and N workers (Figure 1), in which the goal is to compute $Y_1 \triangleq f(X_1), \ldots, Y_K \triangleq f(X_K)$. We denote the *total degree*[3] of the polynomial f by $\deg f$.

In this setting each worker has already stored a fraction of the dataset prior to computation, in a possibly coded manner. Specifically, for $i \in [N]$ (where $[N] \triangleq \{1, \ldots, N\}$), worker i stores $\tilde{X}_i \triangleq g_i(X_1, \ldots, X_K)$, where g_i is a (possibly random) function, refered to as the encoding function of that worker. We restrict our attention to linear encoding schemes[4], which guarantee low encoding complexity and simple implementation.

Each worker $i \in [N]$ computes $\tilde{Y}_i \triangleq f(\tilde{X}_i)$ and returns the result to the master. The master waits for a subset of fastest workers and then decodes $Y_1, \ldots, Y_K$. This procedure must satisfy several additional requirements:

- **Resiliency**, i.e., robustness against stragglers. Formally, the master must be able to obtain the correct values of $Y_1, \ldots, Y_K$ even if up to S workers fail to respond (or respond after the master executes the decoding algorithm), where S is the *resiliency parameter* of the system. A scheme that guarantees resiliency against S stragglers is called *S-resilient*.
- **Security**, i.e., robustness against adversaries. That is, the master must be able to obtain correct values of $Y_1, \ldots, Y_K$ even if up to A workers return arbitrarily erroneous results, where A is the *security parameter* of the system. A scheme that guarantees security against A adversaries is called *A-secure*.
- **Privacy**, i.e., the workers must remain oblivious to the content of the dataset, even if up to T of them collude, where T is the *privacy parameter* of the system. Formally, for every $\mathcal{T} \subseteq [N]$ of size at most T, we must have $I(X; \tilde{X}_{\mathcal{T}}) = 0$, where I is mutual information, $\tilde{X}_{\mathcal{T}}$ represents the collection of the encoded dataset stored at the workers in $\mathcal{T}$, and X is seen as chosen uniformly at random.[5] A scheme which guarantees privacy against T colluding workers is called *T-private*. [6]

More concretely, given any subset of workers that return the computing results (denoted by $\mathcal{K}$), the master computes $(\hat{Y}_1, ..., \hat{Y}_K) = h_{\mathcal{K}}(\{\tilde{Y}_i\}_{i \in \mathcal{K}})$, where each $h_{\mathcal{K}}$ is a deterministic function (or is random but independent of both the encoding functions and input data). We refer to the $h_{\mathcal{K}}$'s as *decoding functions*.[7] We say that a scheme is S-resilient, A-secure, and T-private if the master always returns the correct results (i.e., each $Y_i = \hat{Y}_i$), and all above requirements are satisfied.

Given the above framework, we aim to characterize the region for (S, A, T), such that an S-resilient, A-secure, and T-private scheme can be found, given parameters N, K, and function f, for any sufficiently large field $\mathbb{F}$.

This framework encapsulates many computation tasks of interest, which we highlight as follows.

Linear computation. Consider a scenario where the goal is to compute $A\vec{b}$ for some dataset $A = \{A_i\}_{i=1}^K$ and vector $\vec{b}$, which naturally arises in many machine learning algorithms, such as each iteration of linear regression. Our formulation covers this by letting $\mathbb{V}$ be the space of matrices of certain dimensions over $\mathbb{F}$, $\mathbb{U}$ be the space of vectors of a certain length over $\mathbb{F}$, X_i be A_i, and $f(X_i) = X_i \cdot \vec{b}$ for all $i \in [K]$. Coded computing for such linear computations has also been studied in [9], [12], [21], [28], [29].

Bilinear computation. Another computation task of interest is to evaluate element-wise products $\{A_i \cdot B_i\}_{i=1}^K$ of two lists of matrices $\{A_i\}_{i=1}^K$ and $\{B_i\}_{i=1}^K$. This is the key building block for various algorithms, such as fast distributed matrix multiplication [30]. Our formulation covers this by letting $\mathbb{V}$ be the space of pairs of two matrices of certain dimensions, $\mathbb{U}$ be the space of matrices of dimension which equals that of the product of the pairs of matrices, $X_i = (A_i, B_i)$, and $f(X_i) = A_i \cdot B_i$ for all $i \in [K]$.

General tensor algebra. Beyond bilinear operations, distributed computations of multivariate polynomials of larger degree, such as general tensor algebraic functions (i.e. functions composed of inner products, outer products, and tensor contractions) [31], also arise in practice. A specific example is to compute the coordinate transformation of a third-order tensor field at K locations, where given a list of matrices $\{Q^{(i)}\}_{i=1}^K$ and a list of third-order tensors $\{T^{(i)}\}_{i=1}^K$ with matching dimension on each index, the goal is to compute another list of tensors, denoted by $\{T'^{(i)}\}_{i=1}^K$, of which each entry is defined as $T'^{(i)}_{j'k'\ell'} \triangleq \sum_{j,k,\ell} T^{(i)}_{jk\ell} Q^{(i)}_{jj'} Q^{(i)}_{kk'} Q^{(i)}_{\ell\ell'}$. Our formulation covers all functions within this class by letting $\mathbb{V}$ be the space of input tensors, $\mathbb{U}$ be the space of output tensors, X_i be the inputs, and f be the tensor function. These computations are not studied by state-of-the-art coded computing frameworks.

Gradient computation. Another general class of functions arises from gradient decent algorithms and their variants, which are the workhorse of today's learning tasks [32]. The computation task for this class of functions is to consider one iteration of the gradient decent algorithm, and to evaluate the gradient of the empirical risk $\nabla L_{\mathcal{S}}(h) \triangleq \text{avg}_{z \in \mathcal{S}} \nabla \ell_h(z)$, given a hypothesis $h : \mathbb{R}^d \to \mathbb{R}$, a respective loss function $\ell_h : \mathbb{R}^{d+1} \to \mathbb{R}$, and a training set $\mathcal{S} \subseteq \mathbb{R}^{d+1}$, where d is the number of features. In practice, this computation is carried out by partitioning $\mathcal{S}$ into K subsets $\{\mathcal{S}_i\}_{i=1}^K$ of equal sizes, evaluating the partial gradients $\{\nabla L_{\mathcal{S}_i}(h)\}_{i=1}^K$ distributedly, and computing the final result using $\nabla L_{\mathcal{S}}(h) = \text{avg}_{i \in [K]} \nabla L_{\mathcal{S}_i}(h)$. We present a specific example of applying this computing model to least-squares regression problems in Section VI.

[3]The *total degree* of a polynomial f is the maximum among all the total degrees of its monomials. When discussing finite $\mathbb{F}$, we resort to the canonical representation of polynomials, in which the individual degree within each term is no more than $(|\mathbb{F}| - 1)$.

[4]A formal definition is provided in Section V.

[5]Equivalently, it requires that $\tilde{X}_{\mathcal{T}}$ and X are independent. Under this condition, the input data X still appears uniformly random after the colluding workers learn $\tilde{X}_{\mathcal{T}}$, which guarantees the privacy.

[6]To guarantee that the privacy requirement is well defined, we assume that $\mathbb{F}$ and $\mathbb{V}$ are finite whenever $T > 0$.

[7]Similar to encoding, we also require the decoding function to have low complexity. When there is no adversary ($A = 0$), we restrict our attention to *linear decoding schemes*.

III. Main Results and Prior Works

We now state our main results and discuss their connections with prior works. Our first theorem characterizes the region for (S, A, T) that LCC achieves (i.e., the set of all feasible S-resilient, A-secure, and T-private schemes via LCC as defined in the previos section).

Theorem 1. *Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, LCC provides an S-resilient, A-secure, and T-private scheme for computing $\{f(X_i)\}_{i=1}^K$ for any polynomial f, as long as*

$$(K + T - 1)\deg f + S + 2A + 1 \leq N. \tag{1}$$

Remark 1. To prove Theorem 1, we formally present LCC in Section IV, which achieves the stated resiliency, security, and privacy. The key idea is to encode the input dataset using the well-known Lagrange polynomial. In particular, encoding functions (i.e., g_i's) in LCC amount to evaluations of a Lagrange polynomial of degree $K-1$ at N distinct points. Hence, computations at the workers amount to evaluations of a *composition* of that polynomial with the desired function f. Therefore, inequality (1) may simply be seen as the number of evaluations that are necessary and sufficient in order to interpolate the composed polynomial, which is later evaluated at a certain point to finalize the computation. LCC also has a number of additional properties of interest. First, the proposed encoding is *identical* for all computations f, which allows pre-encoding of the data without knowing the identity of the computing task (i.e., universality). Second, decoding and encoding rely on polynomial interpolation and evaluation, and hence efficient off-the-shelf subroutines can be used.[8]

Remark 2. Besides the coding approach presented to achieve Theorem 1, a variation of LCC can be used to achieve any (S, A, T) as long as $K(S + 2A + \deg\, f \cdot T + 1) \leq N$. This scheme (presented in Appendix D) achieves an improved region when $N < K\deg f - 1$ and $T = 0$, where it recovers the *uncoded repetition* scheme. For brevity, we refer the better of these two scheme as LCC when presenting optimality results (i.e., Theorem 2).

Remark 3. Note that LHS of inequality (1) is independent of the number of workers N, hence the key property of LCC is that adding 1 worker can increase its resilience to stragglers by 1 or its security to malicious servers by $1/2$, while keeping the privacy constraint T the same. Note that using an uncoded replication based approach, to increase the resiliency to stragglers by 1, one needs to essentially repeat *each* computation once more (i.e., requiring K more machines as opposed to 1 machine in LCC). This result essentially extends the well-known optimal scaling of error-correcting codes (i.e., adding one parity can provide robustness against one erasure or $1/2$ error in optimal maximum distance separable codes) to the distributed computing paradigm.

Our next theorem demonstrates the optimality of LCC.

Theorem 2. *LCC achieves the optimal trade-off between resiliency, security, and privacy (i.e., achieving the largest region of (S,A,T)) for any multilinear function f among all computing schemes that uses linear encoding, for all problem scenarios. Moreover, when focusing on the case where no security constraint is imposed, LCC is optimal for any polynomial f among all schemes with additional constraints of linear decoding and sufficiently large (or zero) characteristic of $\mathbb{F}$.*

Remark 4. Theorem 2 is proved in Section V. The main proof idea is to show that any computing strategy that outperforms LCC would violate the decodability requirement, by finding two instances of the computation process where the same intermediate computing results correspond to different output values.

Remark 5. In addition to the result we show in Theorem 2, we can also prove that LCC achieves optimality in terms of the amount of randomness used in data encoding. Specifically, we show in Appendix I that LCC requires injecting the minimum amount of randomness, among all computing schemes that universally achieve the same resiliency-security-privacy tradeoff for all linear functions f.

We conclude this section by discussing several lines of related work in the literature and contrasting them with LCC.

A. LCC vs. Prior Works

The study of coding theoretic techniques for accelerating large scale distributed tasks (a.k.a. *coded computing*) was initiated in [17], [18], [20]. Following works focused largely on matrix-vector and matrix-matrix multiplication (e.g., [21]–[23], [30]), gradient computation in gradient descent algorithms (e.g., [10], [13], [15]), communication reduction via coding (e.g., [33]–[36]), and secure and private computing (e.g., [24], [25]).

LCC recovers several previously studied results as special cases. For example, setting f to be the identity function and $\mathbb{V} = \mathbb{U}$ reduces to the well-studied case of *distributed storage*, in which Theorem 1 is well known (e.g., the Singleton bound [37, Thm. 4.1]). Further, as previously mentioned, f can correspond to matrix-vector and matrix-matrix multiplication, in which the special cases of Theorem 1 are known as well [9], [30].

More importantly, LCC improves and generalizes these works on coded computing in a few aspects: *Generality*–LCC significantly generalizes prior works to go beyond linear and bilinear computations that have so far been the main focus in

[8] A more detailed discussion on the coding complexities of LCC can be found in Appendix B.

	BGW	LCC
Complexity per worker	K	1
Frac. data per worker	1	$1/K$
Randomness	KT	T
Min. num. of workers	$\deg(f)(T+1)$	$\deg(f)(K+T-1)+1$

Table I

COMPARISON BETWEEN BGW BASED DESIGNS AND LCC. THE COMPUTATIONAL COMPLEXITY IS NORMALIZED BY THAT OF EVALUATING f; RANDOMNESS, WHICH REFERS TO THE NUMBER OF RANDOM ENTRIES USED IN ENCODING FUNCTIONS, IS NORMALIZED BY THE LENGTH OF X_i.

this area, and can be applied to arbitrary multivariate polynomial computations that arise in machine learning applications. In fact, many specific computations considered in the past can be seen as special cases of polynomial computation. This includes matrix-vector multiplication, matrix-matrix multiplication, and gradient computation whenever the loss function at hand is a polynomial, or is approximated by one. *Universality*–once the data has been coded, any polynomial up to a certain degree can be computed distributedly via LCC. In other words, data encoding of LCC can be *universally* used for any polynomial computation. This is in stark contrast to previous task specific coding techniques in the literature. Furthermore, workers apply the same computation as if no coding took place; a feature that reduces computational costs, and prevents ordinary servers from carrying the burden of outliers. *Security and Privacy*–other than a handful of works discussed above, straggler mitigation (i.e., resiliency) has been the primary focus of the coded computing literature. This work extends the application of coded computing to secure and private computing for general polynomial computations.

Providing security and privacy for *multiparty computing* (MPC) and Machine Learning systems is an extensively studied topic which addresses a problem setting similar to LCC. To illustrate the significant role of LCC in secure and private computing, let us consider the celebrated BGW MPC scheme [26]. [9]

Given inputs $\{X_i\}_{i=1}^K$, BGW first uses Shamir's scheme [38] to encode the dataset in a privacy-preserving manner as $P_i(z) = X_i + Z_{i,1}z + \ldots + Z_{i,T}z^T$ for every $i \in [K]$, where $Z_{i,j}$'s are i.i.d uniformly random variables and T is the number of colluding workers that should be tolerated. The key distinction between the data encoding of BGW scheme and LCC is that we instead use Lagrange polynomials to encode the data. This results in significant reduction in the amount of randomness needed in data encoding (BGW needs KT $z_{i,j}$'s while as we describe in the next section, LCC only needs T amount of randomness).

The BGW scheme will then store $\{P_i(\alpha_\ell)\}_{i\in[K]}$ to worker ℓ for every $\ell \in [N]$, given some distinct values $\alpha_1, \ldots, \alpha_N$. The computation is then carried out by evaluating f over *all* stored coded data at the nodes. In the LCC scheme, on the other hand, each worker ℓ only needs to store *one* encoded data ($\tilde{X}_\ell$) and compute $f(\tilde{X}_\ell)$. This gives rise to the second key advantage of LCC, which is a factor of K in storage overhead and computation complexity at each worker.

After computation, each worker ℓ in the BGW scheme has essentially evaluated the polynomials $\{f(P_i(z))\}_{i=1}^K$ at $z = \alpha_\ell$, whose degree is at most $\deg(f) \cdot T$. Hence, if no straggler or adversary appears (i.e, $S = A = 0$), the master can recover all required results $f(P_i(0))$'s, through polynomial interpolation, as long as $N \geq \deg(f) \cdot T + 1$ workers participated in the computation[10]. Note that under the same condition, LCC scheme requires $N \geq \deg(f) \cdot (K + T - 1) + 1$ number of workers, which is larger than that of the BGW scheme.

Hence, in overall comparison with the BGW scheme, LCC results in a factor of K reduction in the amount of randomness, storage overhead, and computation complexity, while requiring more workers to guarantee the same level of privacy. This is summarized in Table I.[11]

Recently, [24] has also combined ideas from the BGW scheme and [22] to form *polynomial sharing*, a private coded computation scheme for arbitrary matrix polynomials. However, polynomial sharing inherits the undesired BGW property of performing a communication round for *every* bilinear operation in the polynomial; a feature that drastically increases communication overhead, and is circumvented by the one-shot approach of LCC. *DRACO* [25] is also recently proposed as a secure computation scheme for gradients. Yet, DRACO employs a blackbox approach, i.e., the resulting gradients are encoded rather than the data itself, and the inherent algebraic structure of the gradients is ignored. For this approach, [25] shows that a $2A + 1$ *multiplicative* factor of redundant computations is necessary. In LCC however, the blackbox approach is disregarded in favor of an algebraic one, and consequently, a $2A$ *additive* factor suffices.

LCC has also been recently applied to several applications in which security and privacy in computations are critical. For example, in [39], LCC has been applied to enable a scalable and secure approach to sharding in blockchain systems. Also, in [40], a privacy-preserving approach for machine learning has been developed that leverages LCC to provides substantial speedups over cyrptographic approaches that relay on MPC.

[9]Conventionally, the BGW scheme operates in a multi-round fashion, requiring significantly more communication overhead than one-shot approaches. For simplicity of comparison, we present a modified one-shot version of BGW.

[10]It is also possible to use the conventional multi-round BGW, which only requires $N \geq 2T + 1$ workers to ensure T-privacy. However, multiple rounds of computation and communication ($\Omega(\log \deg(f))$ rounds) are needed, which further increases its communication overhead.

[11]A BGW scheme was also proposed in [26] for secure MPC, however for a substantially different setting. Similarly, a comparison can be made by adapting it to our setting, leading to similar results, which we omit for brevity.

IV. Lagrange Coded Computing

In this Section we prove Theorem 1 by presenting LCC and characterizing the region for (S, A, T) that it achieves.[12] We start with an example to illustrate the key components of LCC.

A. Illustrating Example

Consider the function $f(X_i) = X_i^2$, where input X_i's are $\sqrt{M} \times \sqrt{M}$ square matrices for some square integer M. We demonstrate LCC in the scenario where the input data X is partitioned into $K = 2$ batches X_1 and X_2, and the computing system has $N = 8$ workers. In addition, the suggested scheme is 1-resilient, 1-secure, and 1-private (i.e., achieves $(S, A, T) = (1, 1, 1)$).

The gist of LCC is picking a uniformly random matrix Z, and encoding (X_1, X_2, Z) using a Lagrange interpolation polynomial:[13]

$$u(z) \triangleq X_1 \cdot \frac{(z-2)(z-3)}{(1-2)(1-3)} + X_2 \cdot \frac{(z-1)(z-3)}{(2-1)(2-3)} + Z \cdot \frac{(z-1)(z-2)}{(3-1)(3-2)}.$$

We then fix distinct $\{\alpha_i\}_{i=1}^8$ in $\mathbb{F}$ such that $\{\alpha_i\}_{i=1}^8 \cap [2] = \varnothing$, and let workers $1, \ldots, 8$ store $u(\alpha_1), \ldots, u(\alpha_8)$.

First, note that for every $j \in [8]$, worker j sees $\tilde{X}_j$, a linear combination of X_1 and X_2 that is masked by addition of $\lambda \cdot Z$ for some nonzero $\lambda \in \mathbb{F}_{11}$; since Z is uniformly random, this guarantees perfect privacy for $T = 1$. Next, note that worker j computes $f(\tilde{X}_j) = f(u(\alpha_j))$, which is an evaluation of the composition polynomial $f(u(z))$, whose degree is at most 4, at α_j.

Normally, a polynomial of degree 4 can be interpolated from 5 evaluations at distinct points. However, the presence of $A = 1$ adversary and $S = 1$ straggler requires the master to employ a Reed-Solomon decoder, and have *three* additional evaluations at distinct points (in general, two additional evaluations for every adversary and one for every straggler). Finally, after decoding polynomial $f(u(z))$, the master can obtain $f(X_1)$ and $f(X_2)$ by evaluating it at $z = 1$ and $z = 2$.

B. General Description

Similar to Subsection IV-A, we select any $K + T$ distinct elements $\beta_1, \ldots, \beta_{K+T}$ from $\mathbb{F}$, and find a polynomial $u : \mathbb{F} \to \mathbb{V}$ of degree at most $K + T - 1$ such that $u(\beta_i) = X_i$ for any $i \in [K]$, and $u(\beta_i) = Z_i$ for $i \in \{K + 1, \ldots, K + T\}$, where all Z_i's are chosen uniformly at random from $\mathbb{V}$. This is simply accomplished by letting u be the *Lagrange interpolation polynomial*

$$u(z) \triangleq \sum_{j\in[K]} X_j \cdot \prod_{k\in[K+T]\setminus\{j\}} \frac{z-\beta_k}{\beta_j-\beta_k} + \sum_{j=K+1}^{K+T} Z_j \cdot \prod_{k\in[K+T]\setminus\{j\}} \frac{z-\beta_k}{\beta_j-\beta_k}.$$

We then select N distinct elements $\{\alpha_i\}_{i\in[N]}$ from $\mathbb{F}$ such that $\{\alpha_i\}_{i\in[N]} \cap \{\beta_j\}_{j\in[K]} = \varnothing$ (this requirement is alleviated if $T = 0$), and let $\tilde{X}_i = u(\alpha_i)$ for any $i \in [N]$. That is, the input variables are encoded as

$$\tilde{X}_i = u(\alpha_i) = (X_1, \ldots, X_K, Z_{K+1}, \ldots, Z_{K+T}) \cdot U_i, \tag{2}$$

where $U \in \mathbb{F}_q^{(K+T)\times N}$ is the encoding matrix $U_{i,j} \triangleq \prod_{\ell\in[K+T]\setminus\{i\}} \frac{\alpha_j-\beta_\ell}{\beta_i-\beta_\ell}$, and U_i is its i'th column.[14]

Following the above encoding, each worker i applies f on $\tilde{X}_i$ and sends the result back to the master. Hence, the master obtains $N-S$ evaluations, at most A of which are incorrect, of the polynomial $f(u(z))$. Since $\deg(f(u(z))) \leq \deg(f)\cdot(K+T-1)$, and $N \geq (K + T - 1)\deg(f) + S + 2A + 1$, the master can obtain all coefficients of $f(u(z))$ by applying Reed-Solomon decoding. Having this polynomial, the master evaluates it at β_i for every $i \in [K]$ to obtain $f(u(\beta_i)) = f(X_i)$, and hence we have shown that the above scheme is S-resilient and A-secure.

As for the T-privacy guarantee of the above scheme, our proof relies on the fact that the bottom $T \times N$ submatrix U^{bottom} of U is an MDS matrix (i.e., every $T \times T$ submatrix of U^{bottom} is invertible, see Lemma 2 in the supplementary material). Hence, for a colluding set of workers $\mathcal{T} \subseteq [N]$ of size T, their encoded data $\tilde{X}_\mathcal{T}$ satisfies $\tilde{X}_\mathcal{T} = XU_\mathcal{T}^{top} + ZU_\mathcal{T}^{bottom}$, where $Z \triangleq (Z_{K+1}, \ldots, Z_{K+T})$, and $U_\mathcal{T}^{top} \in \mathbb{F}_q^{K\times T}$, $U_\mathcal{T}^{bottom} \in \mathbb{F}_q^{T\times T}$ are the top and bottom submatrices which correspond to the columns in U that are indexed by $\mathcal{T}$. Now, the fact that any $U_\mathcal{T}^{bottom}$ is invertible implies that the random padding added for these colluding workers is uniformly random, which completely masks the coded data $XU_\mathcal{T}^{top}$. This directly guarantees T-privacy.

[12] For an algorithmic illustration, see Appendix A.

[13] Assume that $\mathbb{F}$ is a finite field with 11 elements.

[14] By selecting the values of α_i's differently, we can recover the uncoded repetition scheme, see Appendix D.

V. Optimality of LCC

In this section, we provide a layout for the proof of optimality for LCC (i.e., Theorem 2). Formally, we define that a *linear encoding function* is one that computes a linear combination of the input variables (and possibly a list of independent uniformly random keys when privacy is taken into account[15]); while a *linear decoding function* computes a linear combination of workers' output. We essentially need to prove that (a) given any multilinear f, any linear encoding scheme that achieves any (S, A, T) requires at least $N \geq (K+T-1)\deg f + S + 2A + 1$ workers when $T > 0$ or $N \geq K\deg\ f - 1$, and $N \geq K(S+2A+1)$ workers in other cases; (b) for a general polynomial f, any scheme that uses linear encoding and decoding requires at least the same number of workers, if the characteristic of $\mathbb{F}$ is 0 or greater than deg f.

The proof rely on the following key lemma, which characterizes the *recovery threshold* of any encoding scheme, defined as the minimum number of workers that the master needs to wait to guarantee decodability.

Lemma 1. *Given any multilinear f, the recovery threshold of any valid linear encoding scheme, denoted by R, satisfies*

$$\begin{aligned} R \geq & R_{\text{LCC}}(N, K, f) \triangleq \\ & \min\{(K-1)\deg f + 1,\ N - \lfloor N/K \rfloor + 1\}. \end{aligned} \tag{3}$$

Moreover, if the encoding scheme is T private, we have $R \geq R_{\text{LCC}}(N, K, f) + T \cdot \deg f$.

The proof of Lemma 1 can be found in Appendix E, by constructing instances of the computation process for any assumed scheme that achieves smaller recovery threshold, and proving that such scheme fails to achieve decodability in these instances. Intuitively, note that the recovery threshold is exactly the difference between N and the number of stragglers that can be tolerated, inequality (3) in fact proves that LCC (described in Section IV and Appendix G) achieves the optimum resiliency, as it exactly achieves the stated recovery threshold. Similarly, one can verify that Lemma 1 essentially states that LCC achieves the optimal tradeoff between resiliency and privacy.

Assuming the correctness of Lemma 1, the two parts of Theorem 2 can be proved as follows. To prove part (a) of the converses, we need to extend Lemma 1 to also take adversaries into account. This is achieved by using an extended concept of Hamming distance, defined in [30] for coded computing. Part (b) requires generalizing Lemma 1 to arbitrary polynomial functions, which is proved by showing that for any f that achieves any (S, T) pair, there exists a multilinear function with the same degree for which a computation scheme can be found to achieves the same requirement. The detailed proofs can be found in Appendices F and G respectively.

VI. Application to Linear Regression and Experiments on AWS EC2

In this section we demonstrate a practical application of LCC in accelerating distributed linear regression, whose gradient computation is a quadratic function of the input dataset, hence matching well the LCC framework. We also experimentally demonstrate its performance gain over state of the arts via experiments on AWS EC2 clusters.

Applying LCC for linear regression. Given a feature matrix $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ containing m data points of d features, and a label vector $\boldsymbol{y} \in \mathbb{R}^m$, a linear regression problem aims to find the weight vector $\boldsymbol{w} \in \mathbb{R}^d$ that minimizes the loss $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||^2$. Gradient descent (GD) solves this problem by iteratively moving the weight along the negative gradient direction, which is in iteration-t computed as $2\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w}^{(t)} - \boldsymbol{y})$.

To run GD distributedly over a system comprising a master node and n worker nodes, we first partition $\boldsymbol{X} = [\boldsymbol{X}_1 \cdots \boldsymbol{X}_n]^\top$ into n sub-matrices. Each worker stores r coded sub-matrices generated from linearly combining $\boldsymbol{X}_j$s, for some parameter $1 \leq r \leq n$. Given the current weight $\boldsymbol{w}$, each worker performs computation using its local storage, and sends the result to the master. Master recovers $\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{w} = \sum_{j=1}^n \boldsymbol{X}_j\boldsymbol{X}_j^\top\boldsymbol{w}$ using the results from a subset of fastest workers.[16] To measure performance of any linear regression scheme, we consider the metric *recovery threshold* (denoted by R), defined as the minimum number of workers the master needs to wait for, to guarantee decodability (i.e., tolerating the remaining stragglers).

We cast this gradient computation to the computing model in Section II, by grouping the sub-matrices into $K = \lceil \frac{n}{r} \rceil$ blocks such that $\boldsymbol{X} = [\bar{\boldsymbol{X}}_1 \cdots \bar{\boldsymbol{X}}_K]^\top$. Then computing $\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{w}$ reduces to computing the sum of a degree-2 polynomial $f(\bar{\boldsymbol{X}}_k) = \bar{\boldsymbol{X}}_k\bar{\boldsymbol{X}}_k^\top\boldsymbol{w}$, evaluated over $\bar{\boldsymbol{X}}_1, \ldots, \bar{\boldsymbol{X}}_K$. Now, we can use LCC to decide on the coded storage as in (2), and achieve a recovery threshold of $R_{\text{LCC}} = 2(K-1)+1 = 2\lceil \frac{n}{r} \rceil - 1$ (Theorem 1).[17]

Comparisons with state of the arts. The conventional uncoded scheme picks $r = 1$, and has each worker j compute $\boldsymbol{X}_j\boldsymbol{X}_j^\top\boldsymbol{w}$. Master needs result from each work, yielding a recovery threshold of $R_{\text{uncoded}} = n$. By redundantly storing/processing $r > 1$ *uncoded* sub-matrices at each worker, the "gradient coding" (GC) methods [10], [14], [15] code across partial gradients computed from uncoded data, and reduce the recovery threshold to $R_{\text{GC}} = n - r + 1$. An alternative "matrix-vector multiplication based" (MVM) approach [17] requires two rounds of computation. In the first round, an intermediate vector $\boldsymbol{z} = \boldsymbol{X}\boldsymbol{w}$ is computed distributedly, which is re-distributed to the workers in the second round for them to collaboratively compute $\boldsymbol{X}^\top\boldsymbol{z}$. Each

[15]This is well defined as we assumed that $\mathbb{V}$ is finite when $T > 0$.

[16]Since the value of $\boldsymbol{X}^\top\boldsymbol{y}$ does not vary across iterations, it only needs to be computed once. We assume that it is available at the master for weight updates.

[17]This recovery threshold is also optimum within a factor of 2, as we proved in Appendix J.

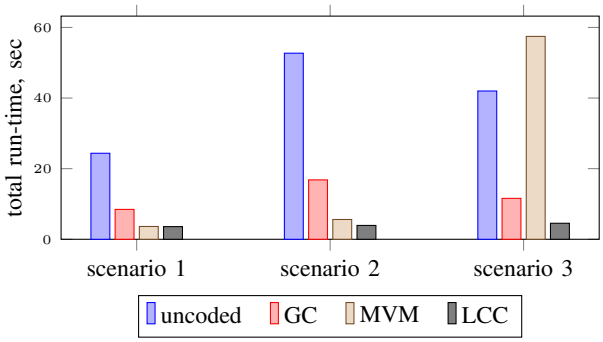

Figure 2. Run-time comparison of LCC with other three schemes: conventional uncoded, GC, and MVM.

worker stores coded data generated using MDS codes from $\boldsymbol{X}$ and $\boldsymbol{X}^\top$ respectively. MVM achieves a recovery threshold of $R_{\text{MVM}} = \lceil \frac{2n}{r} \rceil$ in *each* round, when the storage is evenly split between rounds.

Compared with GC, LCC codes directly on data, and reduces the recovery threshold by about $r/2$ times. While the amount of computation and communication at each worker is the same for GC and LCC, LCC is expected to finish much faster due to its much smaller recovery threshold. Compared with MVM, LCC achieves a smaller recovery threshold than that in each round of MVM (assuming even storage split). While each MVM worker performs less computation in each iteration, it sends two vectors whose sizes are respectively proportional to m and d, whereas each LCC worker only sends one dimension-d vector.

We run linear regression on AWS EC2 using Nesterov's accelerated gradient descent, where all nodes are implemented on `t2.micro` instances. We generate synthetic datasets of m data points, by 1) randomly sampling a true weight $\boldsymbol{w}^*$, 2) randomly sampling each input $\boldsymbol{x}_i$ of d features and computing its output $y_i = \boldsymbol{x}_i^\top \boldsymbol{w}^*$. For each dataset, we run GD for 100 iterations over $n = 40$ workers. We consider different dimensions of input matrix $\boldsymbol{X}$ as listed in the following scenarios.

- Scenario 1 & 2: $(m, d) = (8000, 7000)$.
- Scenario 3: $(m, d) = (160000, 500)$.

We let the system run with naturally occurring stragglers in scenario 1. To mimic the effect of slow/failed workers, we artificially introduce stragglers in scenarios 2 and 3, by imposing a 0.5 seconds delay on each worker with probability 5% in each iteration.

To implement LCC, we set the β_i parameters to $1, ..., \frac{n}{r}$, and the α_i parameters to $0, \ldots, n-1$. To avoid numerical instability due to large entries of the decoding matrix, we can embed input data into a large finite field, and apply LCC in it with exact computations. However in all of our experiments the gradients are calculated correctly without carrying out this step.

Results. For GC and LCC, we optimize the total run-time over r subject to local memory size. For MVM, we further optimize the run-time over the storage assigned between two rounds of matrix-vector multiplications. We plot the measured run-times in Figure 2, and list the detailed breakdowns of all scenarios in Appendix K.

We draw the following conclusions from experiments.

- LCC achieves the least run-time in all scenarios. In particular, LCC speeds up the uncoded scheme by $6.79\times$-$13.43\times$, the GC scheme by 2.36-4.29$\times$, and the MVM scheme by 1.01-12.65$\times$.
- In scenarios 1 & 2 where the number of inputs m is close to the number of features d, LCC achieves a similar performance as MVM. However, when we have much more data points in scenario 3, LCC finishes substantially faster than MVM by as much as $12.65\times$. The main reason for this subpar performance is that MVM requires large amounts of data transfer from workers to the master in the first round and from master to workers in the second round (both are proportional to m). However, the amount of communication from each worker or master is proportional to d for all other schemes, which is much smaller than m in scenario 3.

ACKNOWLEDGEMENT

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575, ONR Award No. N00014-16-1-2189, and CCF-1763673. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. M. Soltanolkotabi is supported by the Packard Fellowship in Science and Engineering, a Sloan Research Fellowship in Mathematics, an NSF-CAREER under award #1846369, the Air Force Office of Scientific Research Young Investigator Program (AFOSR-YIP) under award #FA9550-18-1-0078, an NSF-CIF award #1813877, and a Google faculty research award. Qian Yu is supported by the Google PhD Fellowship.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[2] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[3] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, (Cambridge, MA, USA), pp. 19–27, MIT Press, 2014.

[4] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *Journal of Machine Learning Research*, vol. 17, no. 106, pp. 1–37, 2016.

[5] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, pp. 19–27, 2014.

[6] P. Blanchard, R. Guerraoui, J. Stainer, *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, pp. 118–128, 2017.

[7] R. Cramer, I. B. Damgrd, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. New York, NY, USA: Cambridge University Press, 1st ed., 2015.

[8] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS '08, (Berlin, Heidelberg), pp. 192–206, Springer-Verlag, 2008.

[9] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, pp. 1514–1529, March 2018.

[10] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3368–3376, 2017.

[11] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with ldpc codes," *SysML Conference*, 2018.

[12] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, pp. 5440–5448, 2017.

[13] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," *arXiv preprint arXiv:1710.09990*, 2017.

[14] W. Halbawi, N. A. Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," *CoRR*, vol. abs/1706.05436, 2017.

[15] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic mds codes and expander graphs," *arXiv preprint arXiv:1707.03858*, 2017.

[16] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," *arXiv preprint arXiv:1802.03475*, 2018.

[17] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *NIPS Workshop on Machine Learning Systems*, Dec. 2015.

[18] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," in *Proceedings of the 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 964–971, Sept. 2015.

[19] Q. Yu, S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "How to optimally allocate resources for coded distributed computing?," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, May 2017.

[20] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, 2018.

[21] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *Advances In Neural Information Processing Systems*, pp. 2092–2100, 2016.

[22] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30*, pp. 4406–4416, Curran Associates, Inc., 2017.

[23] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. R. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv preprint arXiv:1801.10292*, 2018.

[24] H. A. Nodehi and M. A. Maddah-Ali, "Limited-sharing multi-party computation for massive matrix operations," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1231–1235, June 2018.

[25] L. Chen, Z. Charles, D. Papailiopoulos, *et al.*, "Draco: Robust distributed training via redundant gradients," *arXiv preprint arXiv:1803.09877*, 2018.

[26] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pp. 1–10, ACM, 1988.

[27] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 19–38, May 2017.

[28] R. Bitar, P. Parag, and S. E. Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *arXiv preprint arXiv:1802.02640*, 2018.

[29] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," *arXiv preprint arXiv:1804.09791*, 2018.

[30] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv preprint arXiv:1801.07487*, 2018.

[31] P. Renteln, *Manifolds, Tensors, and Forms: An Introduction for Mathematicians and Physicists*. Cambridge University Press, 2013.

[32] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[33] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, "Coded terasort," *IPDPSW*, 2017.

[34] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: an alternative trade-off curve," *arXiv preprint arXiv:1705.08966*, 2017.

[35] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr, "Coded computing for distributed graph analytics," *arXiv preprint arXiv:1801.05522*, 2018.

[36] K. Konstantinidis and A. Ramamoorthy, "Leveraging Coding Techniques for Speeding up Distributed Computing," *ArXiv e-prints*, 2018.

[37] R. Roth, *Introduction to coding theory*. Cambridge University Press, 2006.

[38] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, pp. 612–613, Nov. 1979.

[39] S. Li, M. Yu, S. Avestimehr, S. Kannan, and P. Viswanath, "Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously," *arXiv preprint arXiv:1809.10361*, 2018.

[40] J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, "Codedprivateml: A fast and privacy-preserving framework for distributed machine learning," *arXiv preprint arXiv:1902.00641*, 2019.

[41] K. S. Kedlaya and C. Umans, "Fast polynomial factorization and modular composition," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1767–1802, 2011.

[42] E. Berlekamp, "Nonbinary bch decoding (abstr.)," *IEEE Transactions on Information Theory*, vol. 14, pp. 242–242, March 1968.

[43] J. Massey, "Shift-register synthesis and bch decoding," *IEEE Transactions on Information Theory*, vol. 15, pp. 122–127, January 1969.

[44] M. Sudan, "Notes on an efficient solution to the rational function interpolation problem," *Avaliable from http://people.csail.mit.edu/madhu/FT01/notes/rational.ps*, 1999.

[45] M. Rosenblum, "A fast algorithm for rational function approximations," *Avaliable from http://people.csail.mit.edu/madhu/FT01/notes/rosenblum.ps*, 1999.

[46] V. Y. Pan, "Matrix structures of vandermonde and cauchy types and polynomial and rational computations," in *Structured Matrices and Polynomials*, pp. 73–116, Springer, 2001.

[47] W. Huang, *Coding for Security and Reliability in Distributed Systems*. PhD thesis, California Institute of Technology, 2017.

SUPPLEMENTARY MATERIAL

A. Algorithmic Illustration of LCC

Algorithm 1 LCC Encoding (Precomputation)

procedure ENCODE($X_1, X_2, ..., X_K, T$) ▷ Encode inputs variables according to LCC
  generate uniform random variables $Z_{K+1}, ..., Z_{K+T}$
  jointly compute $\tilde{X}_i \leftarrow \sum_{j\in[K]} X_j \cdot \prod_{k\in[K+T]\setminus\{j\}} \frac{\alpha_i-\beta_k}{\beta_j-\beta_k} + \sum_{j=K+1}^{K+T} Z_j \cdot \prod_{k\in[K+T]\setminus\{j\}} \frac{\alpha_i-\beta_k}{\beta_j-\beta_k}$ for $i = 1, 2, ..., N$ using fast polynomial interpolation
  return $\tilde{X}_1, ..., \tilde{X}_N$ ▷ The coded variable assigned to worker i is $\tilde{X}_i$
end procedure

Algorithm 2 Computation Stage

procedure WORKERCOMPUTATION($\tilde{X}$) ▷ Each worker i takes $\tilde{X}_i$ as input
  return $f(\tilde{X})$ ▷ Compute as if no coding is taking place
end procedure

procedure DECODE(S, A) ▷ Executed by master
  wait for a subset of fastest $N - S$ workers
  $\mathcal{N} \leftarrow$ identities of the fastest workers
  $\{\tilde{Y}_i\}_{i\in\mathcal{N}} \leftarrow$ results from the fastest workers
  recover $Y_1, ..., Y_K$ from $\{\tilde{Y}_i\}_{i\in\mathcal{N}}$ using fast interpolation or Reed-Solomon decoding ▷ See Appendix B
  return $Y_1, ..., Y_K$
end procedure

$\beta_1, \ldots, \beta_{K+T}$ and $\alpha_1, ..., \alpha_N$ are global constants in $\mathbb{F}$, satisfying[18]

1) β_i's are distinct,
2) α_i's are distinct,
3) $\{\alpha_i\}_{i\in[N]} \cap \{\beta_j\}_{j\in[K]} = \varnothing$ (this requirement is alleviated if $T = 0$).

B. Coding Complexities of LCC

By exploiting the algebraic structure of LCC, we can find efficient encoding and decoding algorithms with almost linear computational complexities. The encoding of LCC can be viewed as interpolating degree $K + T - 1$ polynomials, and then evaluating them at N points. It is known that both operations only require almost linear complexities: interpolating a polynomial of degree k has a complexity of $O(k \log^2 k \log\log k)$, and evaluating it at any k points requires the same [41]. Hence, the total encoding complexity of LCC is at most $O(N \log^2(K + T) \log\log(K + T) \dim \mathbb{V})$, which is almost linear to the output size of the encoder $O(N \dim \mathbb{V})$.

Similarly, when no security requirement is imposed on the system (i.e., $A = 0$), the decoding of LCC can also be completed using polynomial interpolation and evaluation. An almost linear complexity $O(R \log^2 R \log\log R \dim \mathbb{U})$ can be achieved, where R denotes the recovery threshold.

A less trivial case is to consider the decoding algorithm when $A > 0$, where the goal is essentially to interpolate a polynomial with at most A erroneous input evaluations, or decoding a Reed-Solomon code. An almost linear time complexity can be achieved using additional techniques developed in [42]–[45]. Specifically, the following $2A - 1$ *syndrome variables* can be computed with a complexity of $O((N - S) \log^2(N - S) \log\log(N - S) \dim \mathbb{U})$ using fast algorithms for polynomial evaluation and for transposed-Vandermonde-matrix multiplication [46].

$$S_k \triangleq \sum_{i\in\mathcal{N}} \frac{\tilde{Y}_i \alpha_i^k}{\prod_{j\in\mathcal{N}\setminus\{i\}}(\alpha_i - \alpha_j)} \qquad \forall k \in \{0, 1, ..., 2A - 1\}. \tag{4}$$

According to [42], [43], the location of the errors (i.e., the identities of adversaries in LCC decoding) can be determined given these syndrome variables by computing its rational function approximation. Almost linear time algorithms for this operation are provided in [44], [45], which only requires a complexity of $O(A \log^2 A \log\log A \dim \mathbb{U})$. After identifying the adversaries, the final results can be computed similar to the $A = 0$ case. This approach achieves a total decoding complexity

[18] A variation of LCC is presented in Appendix D, by selecting different values of α_i's.

of $O((N-S)\log^2(N-S)\log\log(N-S)\dim\mathbb{U})$, which is almost linear with respect to the input size of the decoder $O((N-S)\dim\mathbb{U})$.

Finally, note that the adversaries can only affect a *fixed* subset of A workers' results for all entries. This decoding time can be further reduced by computing the final outputs entry-wise: for each iteration, ignore computing results from adversaries identified in earlier steps, and proceed decoding with the rest of the results.

C. The MDS property of U^{bottom}

Lemma 2. *The matrix U^{bottom} is an MDS matrix.*

Proof. First, let $V \in \mathbb{F}^{T\times N}$ be

$$V_{i,j} = \prod_{\ell\in[T]\setminus\{i\}} \frac{\alpha_j - \beta_{\ell+K}}{\beta_{i+K} - \beta_{\ell+K}}.$$

It follows from the resiliency property of LCC that by having $(\tilde{X}_1,\dots,\tilde{X}_N) = (X_1,\dots,X_T)\cdot V$, the master can obtain the values of $X_1,\dots,X_T$ from any T of the $\tilde{X}_i$'s. This is one of the alternative definitions for an MDS code, and hence, V is an MDS matrix.

To show that U^{bottom} is an MDS matrix, it is shown that U^{bottom} can be obtained from V by multiplying rows and columns by nonzero scalars. Let $[T:K] \triangleq \{T+1, T+2, \dots, T+K\}$, and notice that for $(s,r)\in[T]\times[N]$, entry (s,r) of U^{bottom} can be written as

$$\prod_{t\in[K+T]\setminus\{s+K\}} \frac{\alpha_r - \beta_t}{\beta_{s+K} - \beta_t} = \prod_{t\in[K]} \frac{\alpha_r - \beta_t}{\beta_{s+K} - \beta_t} \cdot \prod_{t\in[K:T]\setminus\{s+K\}} \frac{\alpha_r - \beta_t}{\beta_{s+K} - \beta_t}.$$

Hence, U^{bottom} can be written as

$$U^{bottom} = \operatorname{diag}\left(\left(\prod_{t\in[K]} \frac{1}{\beta_{s+K}-\beta_t}\right)_{s\in[T]}\right)\cdot V \cdot \operatorname{diag}\left(\left(\prod_{t\in[K]}(\alpha_r-\beta_t)\right)_{r\in[N]}\right), \tag{5}$$

where V is a $T\times N$ matrix such that

$$V_{i,j} = \prod_{t\in[T]\setminus\{i\}} \frac{\alpha_j - \beta_{t+K}}{\beta_{i+K} - \beta_{t+K}}.$$

Since $\{\beta_t\}_{t=1}^K \cap \{\alpha_r\}_{r=1}^N = \varnothing$, and since all the β_i's are distinct, it follows from (5) that U^{bottom} can be obtained from V by multiplying each row and each column by a nonzero element, and hence U^{bottom} is an MDS matrix as well. □

D. The Uncoded Version of LCC

In Section IV-B, we have described the LCC scheme, which provides an S-resilient, A-secure, and T-private scheme as long as $(K+T-1)\deg f + S + 2A + 1 \le N$. Instead of explicitly following the same construction, a variation of LCC can be made by instead selecting the values of α_i's from the set $\{\beta_j\}_{j\in[K]}$ (not necessarily distinctly).

We refer to this approach as the *uncoded version of LCC*, which essentially recovers the *uncoded repetition* scheme, which simply replicates each X_i onto multiple workers. By replicating every X_i between $\lfloor N/K\rfloor$ and $\lceil N/K\rceil$ times, it can tolerate at most S stragglers and A adversaries, whenever

$$S + 2A \le \lfloor N/K\rfloor - 1, \tag{6}$$

which achieves the optimum resiliency and security when the number of workers is small and no data privacy is required (specifically, $N < K\deg f - 1$ and $T = 0$, see Section V).

When privacy is taken into account (i.e., $T > 0$), an alternative approach in place of repetition is to instead store each input variable using Shamir's secret sharing scheme [38] over $\lfloor N/K\rfloor$ to $\lceil N/K\rceil$ machines. This approach achieves any (S,A,T) tuple whenever $N \ge K(S + 2A + \deg f\cdot T + 1)$. However, it does not improve LCC.

E. Proof of Lemma 1

We start by defining the following notations. For any multilinear function f defined on $\mathbb{V}$ with degree d, let $X_{i,1}, X_{i,2}, ..., X_{i,d}$ denote its d input entries (i.e., $X_i = (X_{i,1}, X_{i,2}, ..., X_{i,d})$ and f is linear with respect to each entry). Let $\mathbb{V}_1, ..., \mathbb{V}_d$ be the vector space that contains the values of the entries. For brevity, we denote $\deg f$ by d in this appendix. We first provide the proof of inequality (3).

Proof of inequality (3). Without loss of generality, we assume both the encoding and decoding functions are deterministic in this proof, as the randomness does not help with decodability.[19] Similar to [30], we define the minimum recovery threshold, denoted by $R^*(N, K, f)$, as the minimum number of workers that the master has to wait to guarantee decodability, among all linear encoding schemes. Then we essentially need to prove that $R^*(N, K, f) \geq R^*_{\text{LCC}}(N, K, f)$, i.e., $R^*(N, K, f) \geq (K-1)d + 1$ when $N \geq Kd - 1$, and $R^*(N, K, f) \geq N - \lfloor N/K \rfloor + 1$ when $N < Kd - 1$.

Obviously $R^*(N, K, f)$ is a non-decreasing function with respect to N. Hence, it suffices to prove that $R^*(N, K, f) \geq N - \lfloor N/K \rfloor + 1$ when $N \leq Kd - 1$. We prove this converse bound by induction.

(a) If $d = 1$, then f is a linear function, and we aim to prove $R^*(N, K, f) \geq N + 1$ for $N \leq K - 1$. This essentially means that no valid computing schemes can be found when $N < K$. Assuming the opposite, suppose we can find a valid computation design using at most $K - 1$ workers, then there is a decoding function that computes all $f(X_i)$'s given the results from these workers.

Because the encoding functions are linear, we can thus find a non-zero vector $(a_1, ..., a_K) \in \mathbb{F}^K$ such that when $X_i = a_i V$ for any $V \in \mathbb{V}$, the coded variable $\tilde{X}_i$ stored by any worker equals the padded random key, which is a constant. This leads to a fixed output from the decoder. On the other hand, because f is assumed to be non-zero, the computing results $\{f(X_i)\}_{i\in[K]}$ is variable for different values of V, which leads to a contradiction. Hence, we have prove the converse bound for $d = 1$.

(b) Suppose we have a matching converse for any multilinear function with $d = d_0$. We now prove the lower bound for any multilinear function f of degree $d_0 + 1$. Similar to part (a), it is easy to prove that $R^*(N, K, f) \geq N + 1$ for $N \leq K - 1$. Hence, we focus on $N \geq K$.

The proof idea is to construct a multilinear function f' with degree d_0 based on function f, and to lower bound the minimum recovery threshold of f using that of f'. More specifically, this is done by showing that given any computation design for function f, a computation design can also be developed for the corresponding f', which achieves a recovery threshold that is related to that of the scheme for f.

In particular, for any non-zero function $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0+1})$, we let f' be a function which takes inputs $X_{i,1}, X_{i,2}, ..., X_{i,d_0}$ and returns a linear map, such that given any $X_{i,1}, X_{i,2}, ..., X_{i,d_0+1}$, we have $f'(X_{i,1}, X_{i,2}, ..., X_{i,d_0})(X_{i,d_0+1}) = f(X_{i,1}, X_{i,2}, ..., X_{i,d_0+1})$. One can verify that f' is a multilinear function with degree d_0, Given parameters K and N, we now develop a computation strategy for f' for a dataset of K inputs and a cluster of $N' \triangleq N - K$ workers, which achieves a recovery threshold of $R^*(N, K, f) - (K - 1)$. We construct this computation strategy based on an encoding strategy of f that achieves the recovery threshold $R^*(N, K, f)$. For brevity, we refer to these two schemes as the f'-scheme and f-scheme respectively.

Because the encoding functions are linear, we consider the encoding matrix, denoted by $G \in \mathbb{F}^{K\times N}$, and defined as the coefficients of the encoding functions $\tilde{X}_i = \sum_{j=1}^{K} X_j G_{ji} + \tilde{z}_i$, where $\tilde{z}_i$ denotes the value of the random key padded to variable $\tilde{X}_i$. Following the same arguments we used in the $d = 1$ case, the left null space of G must be $\{0\}$. Consequently, the rank of G equals K, and we can find a subset $\mathcal{K}$ of K workers such that the corresponding columns of G form a basis of $\mathbb{F}^K$. Hence, we can construct the f'-scheme by letting each of the $N' \triangleq N - K$ workers store the coded version of $(X_{i,1}, X_{i,2}, \ldots, X_{i,d_0})$ that is stored by a unique respective worker in $[N] \setminus \mathcal{K}$ in f-scheme.[20]

Now it suffices to prove that the above construction achieves a recovery threshold of $R^*(N, K, f) - (K - 1)$. Equivalently, we need to prove that given any subset $\mathcal{S}$ of $[N]\backslash\mathcal{K}$ of size $R^*(N, K, f) - (K - 1)$, the values of $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, x)$ for any $i \in [K]$ and $x \in \mathbb{V}$ are decodable from the computing results of workers in $\mathcal{S}$.

We exploit the decodability of the computation design for function f. For any $j \in \mathcal{K}$, the set $\mathcal{S}\cup\mathcal{K}\backslash\{j\}$ has size $R^*(N, K, f)$. Consequently, for any vector $(x_{1,d_0+1}, ..., x_{K,d_0+1}) \in \mathbb{V}^K_{d_0+1}$, we have that $\{f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, x_{i,d_0+1})\}_{i\in[K]}$ is decodable given the results from workers in $\mathcal{S}\cup\mathcal{K}\backslash\{j\}$ computed in f-scheme, if each x_{i,d_0+1} is used as the $(d_0 + 1)$th entree for each input.

Because columns of G with indices in $\mathcal{K}$ form a basis of $\mathbb{F}^K$, we can find values for each input X_{i,d_0+1} such that workers in $\mathcal{K}$ would store 0 for the X_{i,d_0+1} entry in the f-scheme. We denote these values by $\bar{x}_{1,d_0+1}, ..., \bar{x}_{K,d_0+1}$. Note that if these values are taken as inputs, workers in $\mathcal{K}$ would return constant 0 due to the multilinearity of f. Hence, decoding $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, \bar{x}_{i,d_0+1})$ only requires results from workers not in $\mathcal{K}$, i.e., it can be decoded given computing results from workers in $\mathcal{S}$ using the f-scheme. Note that these results can be directly computed from corresponding results in the f'-scheme. We have proved the decodability of $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, x)$ for $x = \bar{x}_{i,d_0+1}$.

[19]Note that this argument requires the assumption that the decoder does not have access to the random keys, as assumed in Section II.

[20]For breivity, in this proof we instead index these $N - K$ workers also using the set $[N] \setminus \mathcal{K}$, following the natural bijection.

Now it remains to prove the decodability of $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, x)$ for each i for general $x \in \mathbb{V}$. For any $j \in \mathcal{K}$, let $\boldsymbol{a}^{(j)} \in \mathbb{F}^K$ be a non-zero vector that is orthogonal to all columns of G with indices in $\mathcal{K}\backslash\{j\}$. If $a_i^{(j)} x + \bar{x}_{i,d_0+1}$ is used for each input X_{i,d_0+1} in the f-scheme, then workers in $\mathcal{K}\backslash\{j\}$ would store 0 for the X_{i,d_0+1} entry, and return constant 0 due to the multilinearity of f. Recall that $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, a_i^{(j)} x + \bar{x}_{i,d_0+1})$ is assumed to be decodable in the f-scheme given results from workers in $\mathcal{S} \cup \mathcal{K}\backslash\{j\}$. Following the same arguments above, one can prove that $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, a_i^{(j)} x + \bar{x}_{i,d_0+1})$ is also decodable using the f'-scheme. Hence, the same applies for $a_i^{(j)} f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, x)$ due to multilinearity of f.

Because columns of G with indices in $\mathcal{K}$ form a basis of $\mathbb{F}^K$, the vectors $\boldsymbol{a}^{(j)}$ for $j \in \mathcal{K}$ also from a basis. Consequently, for any i there is a non-zero $a_i^{(j)}$, and thus $f(X_{i,1}, X_{i,2}, ..., X_{i,d_0}, x)$ is decodable. This completes the proof of decodability.

To summarize, we have essentially proved that $R^*(N, K, f) - (K-1) \geq R^*(N-K, K, f')$. We can verify that the converse bound $R^*(N, K, f) \geq N - \lfloor N/K \rfloor + 1$ under the condition $N \leq Kd - 1$ can be derived given the above result and the induction assumption, for any function f with degree $d_0 + 1$.

(c) Thus, a matching converse holds for any $d \in \mathbb{N}_+$, which proves inequality (3). □

Now we proceed to prove the rest of Lemma 1, explicitly, we aim to prove that the recovery threshold of any T-private encoding scheme is at least $R_{\text{LCC}}(N, K, f) + T \cdot \deg f$. Inequality (3) essentially covers the case for $T = 0$. Hence, we focus on $T > 0$. To simplify the proof, we prove a stronger version of this statement: when $T > 0$, any valid T-private encoding scheme uses at least $N \geq R_{\text{LCC}}(N, K, f) + T \cdot \deg f$ workers. Equivalently, we aim to show that $N \geq (K + T - 1) \deg f + 1$ for any such scheme.

We prove this fact using an inductive approach. To enable an inductive structure, we prove a even stronger converse by considering a more general class of computing tasks and a larger class of encoding schemes, formally stated in the following lemma.

Lemma 3. *Consider a dataset with inputs $X \triangleq (X_1, ..., X_K) \in (\mathbb{F}^d)^K$, and an input vector $\Gamma \triangleq (\Gamma_1, ..., \Gamma_K)$ which belongs to a given subspace of $\mathbb{F}^K$ with dimension $r > 0$; a set of N workers where each can take a coded variable in $\mathbb{F}^{d+1}$ and return the product of its elements; and a computing task where the master aim to recover $Y_i \triangleq X_{i,1} \cdot ... \cdot X_{i,d} \cdot \Gamma_i$. If the inputs entries are encoded separately such that each of the first d entries assigned to each worker are some $T_{\text{X}} > 0$-privately linearly coded version of the corresponding entries of X_i's, and the $(d+1)$th entry assigned to each worker is a T-privately[21] linearly coded version of Γ, moreover, if each Γ_i (as a variable) is non-zero, then any valid computing scheme requires $N \geq (T_{\text{X}} + K - 1)d + T + r$.*

Proof. Lemma 3 is proved by induction with respect to the tuple (d, T, r). Specifically, we prove that (a) Lemma 3 holds when $(d, T, r) = (0, 0, 1)$; (b) If Lemma 3 holds for any $(d, T, r) = (d_0, 0, r_0)$, then it holds when $(d, T, r) = (d_0, 0, r_0 + 1)$; (c) If Lemma 3 holds for any $(d, T, r) = (d_0, 0, r_0)$, then it holds when $(d, T, r) = (d_0, T, r_0)$ for any T; (d) If Lemma 3 holds for any $d = d_0$ and arbitrary values of T and r, then it holds if $(d, T, r) = (d_0 + 1, 0, 1)$. Assuming the correctness of these statements, Lemma 3 directly follows by induction's principle. Now we provide the proof of these statements as follows.

(a). When $(d, T, r) = (0, 0, 1)$, we need to show that at least 1 worker is needed. This directly follows from the decodability requirement, because the master aims to recover a variable, and at least one variable is needed to provide the information.

(b). Assuming that for any $(d, T, r) = (d_0, 0, r_0)$ and any K and T_{X}, any valid computing scheme requires $N \geq (T_{\text{X}} + K - 1)d_0 + r$ workers, we need to prove that for $(d, T, r) = (d_0, 0, r_0 + 1)$, at least $(T_{\text{X}} + K - 1)d_0 + r_0 + 1$ workers are needed. We prove this fact by fixing an arbitrary valid computing scheme for $(d, T, r) = (d_0, 0, r_0 + 1)$. For brevity, let $\tilde{\Gamma}_i$ denotes the coded version of Γ stored at worker i. We consider the following two possible scenarios: (i) there is a worker i such that $\tilde{\Gamma}_i$ is not identical (up to a constant factor) to any variable Γ_j, or (ii) for any worker i, $\tilde{\Gamma}_i$ is identical (up to a constant factor) to some Γ_j.

For case (i), similar to the ideas we used to prove inequality (3), it suffices to show that if the given computing scheme uses N workers, we can construct another computation scheme achieving the same T_X, for a different computing task with parameters $d = d_0$ and $r = r_0$, using at most $N - 1$ workers.

Recall that we assumed that there is a worker i, such that $\tilde{\Gamma}_i$ is not identical (up to a constant factor) to any Γ_j. We can always restrict the value of Γ to a subspace with dimension r_0, such that $\tilde{\Gamma}_i$ becomes a constant 0. After this operation, from the computation results of the rest $N - 1$ workers, the master can recover a computing function with $r = r_0$ and non-zero Γ_j's, which provides the needed computing scheme.

For case (ii), because each Γ_j is assumed to be non-zero, we can partition the set of indices j into distinct subsets, such that any j and j' are in the same subset iff Γ_j is a constant multiple of $\Gamma_{j'}$. We denote these subsets by $\mathcal{J}_1, ..., \mathcal{J}_m$. Moreover, for any $k \in [m]$, let $\mathcal{I}_k$ denote the subset of indices i such that $\tilde{\Gamma}_i$ is identical (up to a constant factor) to Γ_j for j in $\mathcal{J}_k$.

Now for any $k \in [m]$, we can restrict the value of Γ to a subspace with dimension r_0, such that Γ_j is zero for any $j \in \mathcal{J}_k$. After applying this operation, from the computation results of workers in $[N]\backslash\mathcal{I}_k$, the master can recover a computing function with $r = r_0$, where $K' = K - |\mathcal{J}_k|$ sub-functions has non-zero Γ_j's. By applying the induction assumption on this provided

[21]For this lemma, we assume that no padded random variable is used for a 0-private encoding scheme.

computing scheme, we have $N-|\mathcal{I}_k| \geq (T_{\mathrm{X}}+K-|\mathcal{J}_k|-1)d_0+r_0$. By taking the summation of the this inequality over $k\in[m]$, we have

$$Nm-\sum_{k=1}^{m}|\mathcal{I}_k| \geq (T_{\mathrm{X}}m+Km-K-m)d_0+r_0m. \tag{7}$$

Recall that for any worker i, $\tilde{\Gamma}_i$ is identical (up to a constant factor) to some Γ_j, we have $\cup_{k\in[m]}\mathcal{I}_k=[N]$. Thus, $\sum_k|\mathcal{I}_k|\geq N$. Consequently, inequality (7) implies that

$$Nm-N \geq (T_{\mathrm{X}}m+Km-K-m)d_0+r_0m. \tag{8}$$

Note that $r_0+1>1$, which implies that at least two Γ_j's are not identical up to a constant factor. Hence, $m-1>0$, and (8) is equivalently

$$N \geq \frac{(T_{\mathrm{X}}m+Km-K-m)d_0+r_0m}{m-1} \tag{9}$$

$$= (T_{\mathrm{X}}+K-1)d_0+r_0+((T_{\mathrm{X}}-1)d_0+r_0)\frac{1}{m-1}. \tag{10}$$

Since T_{X} and r_0 are both positive, we have $(T_{\mathrm{X}}-1)d_0+r_0>0$. Consequently, $((T_{\mathrm{X}}-1)d_0+r_0)\frac{1}{m-1}>0$, and we have

$$N \geq (T_{\mathrm{X}}+K-1)d_0+r_0+1, \tag{11}$$

which proves the induction statement.

(c). Assuming that for any $(d,T,r)=(d_0,0,r_0)$, any valid computing scheme requires $N\geq(T_{\mathrm{X}}+K-1)d_0+r_0$ workers, we need to prove that for $(d,T,r)=(d_0,T_0,r_0)$, $N\geq(T_{\mathrm{X}}+K-1)d_0+T_0+r_0$. Equivalently, we aim to show that for any $T_0>0$, in order to provide T_0-privacy to the d_0+1th entry, T_0 extra worker is needed. Similar to the earlier steps, we consider an arbitrary valid computing scheme for $(d,T,r)=(d_0,T_0,r_0)$ that uses N workers. We aim to construct a new scheme for $(d,T,r)=(d_0,0,r_0)$, for the same computation task and the same T_{X}, which uses at most $N-T_0$ workers.

Recall that if an encoding scheme is T_0 private, then given any subset of at most T_0 workers, denoted by $\mathcal{T}$, we have $I(\Gamma;\tilde{\Gamma}_{\mathcal{T}})=0$. Consequently, conditioned on $\tilde{\Gamma}_{\mathcal{T}}=0$, the entropy of the variable Γ remains unchanged. This indicates that Γ can be any possible value when $\tilde{\Gamma}_{\mathcal{T}}=0$. Hence, we can let the values of the padded random variables be some linear combinations of the elements of Γ, such that worker in $\mathcal{T}$ returns constant 0.

Now we construct an encoding scheme as follows. Firstly it is easy to show that when the master aims to recover a non-constant function, at least T_0+1 workers are needed to provide non-zero information regarding the inputs. Hence, we can arbitrarily select a subset of T_0 workers, denoted by $\mathcal{T}$. As we have proved, we can find fix the values of the padded random variables such that $\tilde{\Gamma}_{\mathcal{T}}=0$. Due to multilinearity of the computing task, these workers in $\mathcal{T}$ also returns constant 0. Conditioned on these values, the decoder essentially computes the final output only based on the rest $N-T_0$ workers, which provides the needed computing scheme. Moreover, as we have proved that the values of the padded random variables can be chosen to be some linear combinations of the elements of Γ, our obtained computing scheme encodes Γ linearly. This completes the proof for the induction statement.

(d). Assuming that for any $d=d_0$ and arbitrary values of T and r, any valid computing scheme requires $N\geq(T_{\mathrm{X}}+K-1)d_0+T+r$ workers, we need to prove that for $(d,T,r)=(d_0+1,0,1)$, $N\geq(T_{\mathrm{X}}+K-1)(d_0+1)+1$. Observing that for any computing task with $r=1$, by fixing an non-zero Γ, it essentially computes K functions where each multiplies d_0 variables. Moreover, for each function, by viewing the first (d_0-1) entries as a vector X_i' and by viewing the last entry as a scalar Γ_i', it essentially recovers the case where the parameter d is reduced by 1, K remain unchanged, and r equals K. By adapting any computing scheme in the same way, we have T_{X} remain unchanged, and T becomes T_{X}. Then by induction assumption, any computing scheme for $(d,T,r)=(d_0+1,0,1)$ requires at least $(T_{\mathrm{X}}+K-1)d_0+T_{\mathrm{X}}+K=(T_{\mathrm{X}}+K-1)(d_0+1)+1$ workers. □

Remark 6. Using exactly the same arguments, Lemma 3 can be extended to the case where the entries of X are encoded under different privacy requirements. Specifically, if the ith entry is T_i-privately encoded, then at least $\sum_{i=1}^{d}T_i+(K-1)d+T+r$ worker is needed. Lemma 3 and this extended version are both tight, in the sense for any parameter values of d, K and r, there are computing tasks where a computing scheme that uses the matching number of workers can be found, using constructions similar to the Lagrange coded computing.

Now using Lemma 3, we complete the proof of Lemma 1 for $T>0$. Similar to the proof ideas for inequality (3) part (a), we consider any multilinear function f with degree d, and we find constant vectors $V_1,...,V_d$, such that $f(V_1,...,V_d)$ is non-zero. Then by restricting the input variables to be constant multiples of $V_1,...,V_d$, this computing task reduces to multiplying d scalars, given K inputs. As stated in Lemma 3 and discussed in part (d) of its induction proof, such computation requires $(T+K-1)d+1$ workers. This completes the proof of Lemma 1.

F. Optimality on the Resiliency-Security-Privacy Tradeoff for Multilinear Functions

In this appendix, we prove the first part of Theorem 2 using Lemma 1. Specifically, we aim to prove that LCC achieves the optimal trade-off between resiliency, security, and privacy for any multilinear function f. By comparing Lemma 1 and the achievability result presented in Theorem 1 and Appendix D, we essentially need to show that for any linear encoding scheme that can tolerates A adversaries and S stragglers, it can also tolerate $S + 2A$ stragglers.

This converse can be proved by connecting the straggler mitigation problem and the adversary tolerance problem using the extended concept of Hamming distance for coded computing, which is defined in [30]. Specifically, given any (possibly random) encoding scheme, its hamming distance is defined as the minimum integer, denoted by d, such that for any two instances of input X whose outputs Y are different, and for any two possible realizations of the N encoding functions, the computing results given the encoded version of these two inputs, using the two lists of encoding functions respectively, differs for at least d workers.

It was shown in [30] that this hamming distance behaves similar to its classical counter part: an encoding scheme is S-resilient and A-secure whenever $S + 2A \leq d - 1$. Hence, for any encoding scheme that is A-secure and S-reselient, it has a hamming distance of at least $S + 2A + 1$. Consequently it can tolerate $S + 2A$ stragglers. Combining the above and Lemma 1, we have completed the proof.

G. Optimality on the Resiliency-Privacy Tradeoff for General Multivariate Polynomials

In this appendix, we prove the second part of Theorem 2 using Lemma 1. Specifically, we aim to prove that LCC achieves the optimal trade-off between resiliency and privacy, for general multivariate polynomial f. The proof is carried out by showing that for any function f that allows S-resilient T-private designs, there exists a multilinear function with the same degree for which a computation scheme can be found that achieves the same requirement.

Specifically, given any function f with degree d, we aim to provide an explicit construction of an multilinear function, denoted by f', which achieves the same requirements. The construction satisfies certain properties to ensure this fact. Both the construction and the properties are formally stated in the following lemma (which is proved in Appendix H):

Lemma 4. *Given any function f of degree d, let f' be a map from $\mathbb{V}^d \to \mathbb{U}$ such that $f'(Z_1, ..., Z_d) = \sum_{\mathcal{S}\subseteq[d]} (-1)^{|\mathcal{S}|} f(\sum_{j\in\mathcal{S}} Z_j)$ for any $\{Z_j\}_{j\in[d]} \in \mathbb{V}^d$. Then f' is multilinear with respect to the d inputs. Moreover, if the characteristic of the base field $\mathbb{F}$ is 0 or greater than d, then f' is non-zero.*

Assuming the correctness of Lemma 4, it suffices to prove that f' enables computation designs that tolerates at least the same number of stragglers, and provides at least the same level of data privacy, compared to that of f. We prove this fact by constructing such computing schemes for f' given any design for f.

Note that f' is defined as a linear combination of functions $f(\sum_{j\in\mathcal{S}} Z_j)$, each of which is a composition of a linear map and f. Given the linearity of the encoding design, any computation scheme of f can be directly applied to any of these functions, achieving the same resiliency and privacy requirements. Since the decoding functions are linear, the same scheme also applies to linear combinations of them, which includes f'. Hence, the resiliency-privacy tradeoff achievable for f can also be achieved by f'. This concludes the proof.

H. Proof of Lemma 4

We first prove that f' is multilinear with respect to the d inputs. Recall that by definition, f is a linear combination of monomials, and f' is constructed based on f through a linear operation. By exploiting the commutativity of these these two linear relations, we only need to show individually that each monomial in f is transformed into a multilinear function.

More specifically, let f be the sum of monomials $h_k \triangleq U_k \cdot \prod_{\ell=1}^{d_k} h_{k,\ell}(\cdot)$ where k belongs to a finite set, $U_k \in \mathbb{U}$, $d_k \in \{0, 1, ..., d\}$, and each $h_{k,\ell}$ is a linear map from $\mathbb{V}$ to $\mathbb{F}$. Let h'_k denotes the contribution of h_k in f', then for any $Z = (Z_1, ..., Z_d) \in \mathbb{V}^d$ we have

$$\begin{aligned} h'_k(Z) &= \sum_{\mathcal{S}\subseteq[d]} (-1)^{|\mathcal{S}|} h_k \left(\sum_{j\in\mathcal{S}} Z_j \right) \\ &= \sum_{\mathcal{S}\subseteq[d]} (-1)^{|\mathcal{S}|} U_k \cdot \prod_{\ell=1}^{d_k} h_{k,\ell} \left(\sum_{j\in\mathcal{S}} Z_j \right). \end{aligned} \tag{12}$$

By utilizing the linearity of each $h_{k,\ell}$, we can write h'_k as

$$\begin{aligned} h'_k(Z) &= U_k \cdot \sum_{\mathcal{S}\subseteq[d]} (-1)^{|\mathcal{S}|} \prod_{\ell=1}^{d_k} \sum_{j\in\mathcal{S}} h_{k,\ell}\left(Z_j\right) \\ &= U_k \cdot \sum_{\mathcal{S}\subseteq[d]} (-1)^{|\mathcal{S}|} \prod_{\ell=1}^{d_k} \sum_{j=1}^{d} \mathbb{1}(j\in\mathcal{S}) \cdot h_{k,\ell}\left(Z_j\right) \end{aligned} \tag{13}$$

Then by viewing each subset $\mathcal{S}$ of $[d]$ as a map from $[d]$ to $\{0,1\}$, we have[22]

$$\begin{aligned} h'_k(Z) &= U_k \sum_{\boldsymbol{s}\in\{0,1\}^d} \left(\prod_{m=1}^{d} (-1)^{s_m}\right) \\ &\qquad \cdot \prod_{\ell=1}^{d_k} \sum_{j=1}^{d} s_j \cdot h_{k,\ell}\left(Z_j\right) \\ &= U_k \sum_{\boldsymbol{j}\in[d]^{d_k}} \sum_{\boldsymbol{s}\in\{0,1\}^d} \left(\prod_{m=1}^{d} (-1)^{s_m}\right) \\ &\qquad \cdot \prod_{\ell=1}^{d_k} (s_{j_\ell} \cdot h_{k,\ell}\left(Z_{j_\ell}\right)). \end{aligned} \tag{14}$$

Note that the product $\prod_{\ell=1}^{d_k} s_{j_\ell}$ can be alternatively written as $\prod_{m=1}^{d} s_m^{\#(m \text{ in } \boldsymbol{j})}$, where $\#(m \text{ in } \boldsymbol{j})$ denotes the number of elements in $\boldsymbol{j}$ that equals m. Hence

$$\begin{aligned} h'_k(Z) &= U_k \cdot \sum_{\boldsymbol{j}\in[d]^{d_k}} \sum_{\boldsymbol{s}\in\{0,1\}^d} \left(\prod_{m=1}^{d} \left((-1)^{s_m} s_m^{\#(m \text{ in } \boldsymbol{j})}\right)\right) \\ &\qquad \cdot \prod_{\ell=1}^{d_k} h_{k,\ell}(Z_{j_\ell}) \\ &= U_k \cdot \sum_{\boldsymbol{j}\in[d]^{d_k}} \left(\prod_{m=1}^{d} \sum_{s\in\{0,1\}} (-1)^s s^{\#(m \text{ in } \boldsymbol{j})}\right) \\ &\qquad \cdot \prod_{\ell=1}^{d_k} h_{k,\ell}(Z_{j_\ell}). \end{aligned} \tag{15}$$

The sum $\sum_{s\in\{0,1\}} (-1)^s s^{\#(m \text{ in } \boldsymbol{j})}$ is non-zero only if m appears in $\boldsymbol{j}$. Consequently, among all terms that appear in (15), only the ones with degree $d_k = d$ and distinct elements in $\boldsymbol{j}$ have non-zero contribution. More specifically, [23]

$$h'_k(Z) = (-1)^d \cdot \mathbb{1}(d_k = d) \cdot U_k \cdot \sum_{g\in S_d} \prod_{j=1}^{d} h_{k,g(j)}\left(Z_j\right). \tag{16}$$

Recall that f' is a linear combination of h'_k's. Consequently, it is a multilinear function.

Now we prove that f' is non-zero. From equation (16), we can show that when all the elements Z_j's are identical, $f'(Z)$ equals the evaluation of the highest degree terms of f multiplied by a constant $(-1)^d d!$ with Z_j as the input for any j. Given that the highest degree terms can not be zero, and $(-1)^d d!$ is non-zero as long as the characteristic of the field $\mathbb{F}$ is greater than d, we proved that f' is non-zero.

I. Optimality in randomness

In this appendix, we prove the optimality of LCC in terms of the amount of randomness needed in data encoding, which is formally stated in the following theorem.

[22]Here we define $0^0 = 1$.

[23]Here S_d denotes the symmetric group of degree d.

Theorem 3. *(Optimal randomness) Any linear encoding scheme that universally achieves a same tradeoff point specified in Theorem 1 for all linear functions f (i.e., (S, A, T) such that $K + T + S + 2A = N$) must use an amount of randomness no less than that of LCC.*

Proof. The proof is taken almost verbatim from [47], Chapter 3. In what follows, an $(n, k, r, z)_{\mathbb{F}_q^t}$ *secure RAID scheme* is a storage scheme over $\mathbb{F}_q^t$ (where $\mathbb{F}_q$ is a field with q elements) in which k message symbols are coded into n storage servers, such that the k message symbols are reconstructible from any $n - r$ servers, and any z servers are information theoretically oblivious to the message symbols. Further, such a scheme is assumed to use v random entries as keys, and by [47], Proposition 3.1.1, must satisfy $n - r \geq k + z$.

Theorem 4. *[47], Theorem 3.2.1. A linear rate-optimal $(n, k, r, z)_{\mathbb{F}_q^t}$ secure RAID scheme uses at least zt keys over $\mathbb{F}_q$ (i.e., $v \geq z$).*

Clearly, in our scenario $\mathbb{V}$ can be seen as $\mathbb{F}_q^{\dim \mathbb{V}}$ for some q. Further, by setting $N = n$, $T = z$, and $t = \dim \mathbb{V}$, it follows from Theorem 4 that any encoding scheme which guarantees information theoretic privacy against sets of T colluding workers must use at least T random entries $\{Z_i\}_{i \in [T]}$. □

J. Optimality of LCC for Linear Regression

In this section, we prove that the proposed LCC scheme achieves the minimum possible recovery threshold R^* to within a factor of 2, for the linear regression problem discussed in Section 6.

As the first step, we prove a lower bound on R^* for linear regression. More specifically, we show that for any coded computation scheme, the master always needs to wait for at least $\lceil \frac{n}{r} \rceil$ workers to be able to decode the final result, i.e., $R^* \geq \lceil \frac{n}{r} \rceil$. Before starting the proof, we first note that since here we consider a more general scenario where workers can compute *any* function on locally stored coded sub-matrices (not necessarily matrix-matrix multiplication), the converse result in Theorem 2 no longer holds.

To prove the lower bound, it is equivalent to show that, for any coded computation scheme and any subset $\mathcal{N}$ of workers, if the master can recover $\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}$ given the results from workers in $\mathcal{N}$, then we must have $|\mathcal{N}| \geq \lceil \frac{n}{r} \rceil$. Suppose the condition in the above statement holds, then we can find encoding, computation, and decoding functions such that for any possible values of $\boldsymbol{X}$ and $\boldsymbol{w}$, the composition of these functions returns the correct output.

Note that within a GD iteration, each worker performs its local computation only based on its locally stored coded sub-matrices and the weight vector $\boldsymbol{w}$. Hence, if the master can decode the final output from the results of the workers in a subset $\mathcal{N}$, then the composition of the decoding function and the computation functions of these workers essentially computes $\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}$, using only the coded sub-matrices stored at these workers and the vector $\boldsymbol{w}$. Hence, if any class of input values $\boldsymbol{X}$ gives the same coded sub-matrices for each worker in $\mathcal{N}$, then the product $\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}$ must also be the same given any $\boldsymbol{w}$.

Now we consider the class of input matrices $\boldsymbol{X}$ such that all coded sub-matrices stored at workers in $\mathcal{N}$ equal the values of the corresponding coded sub-matrices when $\boldsymbol{X}$ is zero. Since $\mathbf{0}^\top \mathbf{0}\boldsymbol{w}$ is zero for any $\boldsymbol{w}$, $\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}$ must also be zero for all matrices $\boldsymbol{X}$ in this class and any $\boldsymbol{w}$. However, for real matrices $\boldsymbol{X} = \mathbf{0}$ is the only solution to that condition. Thus, zero matrix must be the only input matrix that belongs to this class.

Recall that all the encoding functions are assumed to be linear. We consider the collection of all encoding functions that are used by workers in $\mathcal{N}$, which is also a linear map. As we have just proved, the kernel of this linear map is $\{\mathbf{0}\}$. Hence, its rank must be at least the dimension of the input matrix, which is dm. On the other hand, its rank is upper bounded by the dimension of the output, where each encoding function from a worker contributes at most $\frac{rdm}{n}$. Consequently, the number of workers in $\mathcal{N}$ must be at least $\lceil \frac{n}{r} \rceil$ to provide sufficient rank to support the computation.

Having proved that $R^* \geq \lceil \frac{n}{r} \rceil$, the factor of two characterization of LCC directly follows since $R^* \leq R_{\text{LCC}} = 2\lceil \frac{n}{r} \rceil - 1 < 2\lceil \frac{n}{r} \rceil \leq 2R^*$.

Note that the converse bound proved above applies to the most general computation model, i.e., there are no assumptions made on the encoding functions or the functions that each worker computes. If additional requirements are taken into account, we can show that LCC achieves the exact optimum recovery threshold (e.g., see [30]).

K. Complete Experimental Results

In this section, we present the complete experimental results using the LCC scheme proposed in the paper, the gradient coding (GC) scheme [10] (the cyclic repetition scheme), the matrix-vector multiplication based (MVM) scheme [17], and the uncoded scheme for which there is no data redundancy across workers, measured from running linear regression on Amazon EC2 clusters.

In particular, experiments are performed for the following 3 scenarios.

- Scenario 1 & 2: # of input data point $m = 8000$, # of features $d = 7000$.
- Scenario 3: # of input data point $m = 160000$, # of features $d = 500$.

In scenarios 2 and 3, we artificially introduce stragglers by imposing a 0.5 seconds delay on each worker with probability 5% in each iteration.

We list the detailed breakdowns of the run-times in 3 experiment scenarios in Tables II, III, and IV respectively. In particular, the computation (comp.) time is measured as the summation of the maximum local processing time among all non-straggling workers, over 100 iterations. The communication (comm.) time is computed as the difference between the total run-time and the computation time.

Table II
BREAKDOWNS OF THE RUN-TIMES IN SCENARIO ONE.

schemes	# batches/ worker (r)	recovery threshold	comm. time	comp. time	total run-time
uncoded	1	40	24.125 s	0.237 s	24.362 s
GC	10	31	6.033 s	2.431 s	8.464 s
MVM Rd. 1	5	8	1.245 s	0.561 s	1.806 s
MVM Rd. 2	5	8	1.340 s	0.480 s	1.820 s
MVM total	10	-	2.585 s	1.041 s	3.626 s
LCC	10	7	1.719 s	1.868 s	3.587 s

Table III
BREAKDOWNS OF THE RUN-TIMES IN SCENARIO TWO.

schemes	# batches/ worker (r)	recovery threshold	comm. time	comp. time	total run-time
uncoded	1	40	7.928 s	44.772 s	52.700 s
GC	10	31	14.42 s	2.401 s	16.821 s
MVM Rd. 1	5	8	2.254 s	0.475 s	2.729 s
MVM Rd. 2	5	8	2.292 s	0.586 s	2.878 s
MVM total	10	-	4.546 s	1.061 s	5.607 s
LCC	10	7	2.019 s	1.906 s	3.925 s

Table IV
BREAKDOWNS OF THE RUN-TIMES IN SCENARIO THREE.

schemes	# batches/ worker (r)	recovery threshold	comm. time	comp. time	total run-time
uncoded	1	40	0.229 s	41.765 s	41.994 s
GC	10	31	8.627 s	2.962 s	11.589 s
MVM Rd. 1	5	8	3.807 s	0.664 s	4.471 s
MVM Rd. 2	5	8	52.232 s	0.754 s	52.986 s
MVM total	10	-	56.039 s	1.418 s	57.457 s
LCC	10	7	1.962 s	2.597 s	4.541 s