

Low-Bandwidth Recovery of Linear Functions of Reed-Solomon-Encoded Data

Noah Shetty ✉

Stanford University, CA, USA

Mary Wootters ✉

Stanford University, CA, USA

Abstract

We study the problem of efficiently computing on encoded data. More specifically, we study the question of *low-bandwidth* computation of functions $F : \mathbb{F}^k \rightarrow \mathbb{F}$ of some data $\mathbf{x} \in \mathbb{F}^k$, given access to an encoding $\mathbf{c} \in \mathbb{F}^n$ of $\mathbf{x}$ under an error correcting code. In our model – relevant in distributed storage, distributed computation and secret sharing – each symbol of $\mathbf{c}$ is held by a different party, and we aim to minimize the total amount of information downloaded from each party in order to compute $F(\mathbf{x})$. Special cases of this problem have arisen in several domains, and we believe that it is fruitful to study this problem in generality.

Our main result is a low-bandwidth scheme to compute linear functions for Reed-Solomon codes, even in the presence of erasures. More precisely, let $\varepsilon > 0$ and let $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ be a full-length Reed-Solomon code of rate $1 - \varepsilon$ over a field $\mathbb{F}$ with constant characteristic. For any $\gamma \in [0, \varepsilon)$, our scheme can compute any linear function $F(\mathbf{x})$ given access to any $(1 - \gamma)$ -fraction of the symbols of $\mathcal{C}(\mathbf{x})$, with download bandwidth $O(n/(\varepsilon - \gamma))$ bits. In contrast, the naive scheme that involves reconstructing the data $\mathbf{x}$ and then computing $F(\mathbf{x})$ uses $\Theta(n \log n)$ bits. Our scheme has applications in distributed storage, coded computation, and homomorphic secret sharing.

2012 ACM Subject Classification Mathematics of computing → Coding theory

Keywords and phrases Reed-Solomon Codes, Regenerating Codes, Coded Computation

Digital Object Identifier 10.4230/LIPIcs.ITCS.2022.117

Related Version *Full Version:* <https://arxiv.org/abs/2107.11847>

Funding *Noah Shetty:* Supported in part by NSF DGE-1656518.

Mary Wootters: Research partially supported by NSF Grants CCF-1844628 and CCF-BSF-1814629, and by a Sloan Research Fellowship.

Acknowledgements We thank Yuval Ishai for helpful conversations, and in particular for suggesting the approach in Remark 7. We also thank Ravi Vakil for helpful conversations.

1 Introduction

Suppose that we would like to store some data $\mathbf{x} \in \mathbb{F}^k$ on a distributed storage system consisting of n nodes, where $n \geq k$. (Here and for the rest of the paper, $\mathbb{F}$ denotes some finite field). Since node failure is a possibility, we may protect the data with an *error correcting code* $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ as follows. We encode $\mathbf{x}$ as a *codeword* $\mathbf{c} = \mathcal{C}(\mathbf{x}) \in \mathbb{F}^n$, and for $i = 1, \dots, n$, we send the symbol c_i to the i 'th storage node. If $\mathcal{C}$ is a *Maximum Distance Separable* (MDS) code – meaning that any k symbols of the codeword $\mathbf{c}$ are sufficient to recover the original data $\mathbf{x}$ – then the system can tolerate $n - k$ node failures without losing any of the original data. Encoding with an MDS code (such as a Reed-Solomon code, see Definition 4 below) is common in distributed storage: for example, Reed-Solomon codes are built into HDFS [15] and Ceph [1].



© Noah Shetty and Mary Wootters;

licensed under Creative Commons License CC-BY 4.0

13th Innovations in Theoretical Computer Science Conference (ITCS 2022).

Editor: Mark Braverman; Article No. 117; pp. 117:1–117:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Given data $\mathbf{x} \in \mathbb{F}^k$ encoded and stored with an MDS code as described above, suppose that we would like to compute a function $F(\mathbf{x})$ of the data, where $F : \mathbb{F}^k \rightarrow \mathbb{F}$. One scheme (which we will refer to as the *naive scheme*) is to contact any k of the nodes, download their data, recover $\mathbf{x}$, and compute $F(\mathbf{x})$. This requires downloading k field symbols, or $k \log |\mathbb{F}|$ bits. We call the amount of downloaded information the *bandwidth* of the scheme. Given that $F(\mathbf{x})$ is only one field symbol, or $\log |\mathbb{F}|$ bits, the naive scheme seems wasteful in terms of bandwidth.

Our motivating question is whether we can compute $F(\mathbf{x})$ with less bandwidth. That is, *when is it possible to do communication-efficient computation on top of encoded data?*

In this paper, we introduce a new notion, *low-bandwidth function evaluation*, in order to make this question precise. Our main result is a low-bandwidth function evaluation scheme for the ubiquitous family of *Reed-Solomon Codes*, and for the useful family of *linear functions* $F : \mathbb{F}^k \rightarrow \mathbb{F}$.

1.1 Low-Bandwidth Function Evaluation

A low-bandwidth evaluation scheme for a code $\mathcal{C}$ and a collection of functions $\mathcal{F}$ allows us to compute functions in $\mathcal{F}$ in a communication-efficient way on data encoded with $\mathcal{C}$, even when a set $\mathcal{I} \subset [n]$ of symbols are unavailable (e.g., the corresponding nodes have failed). More precisely, we have the following definition. Below, and throughout the paper, we use bold letters like $\mathbf{c}$ to denote vectors, and we use c_i to denote the i 'th entry of $\mathbf{c}$.

► **Definition 1** (Low-Bandwidth Function Evaluation). *Let $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ be a code. Let $\mathcal{F}$ be a class of functions $F : \mathbb{F}^k \rightarrow \mathbb{F}$. Let $b \geq 0$. We say that there is an evaluation scheme for $\mathcal{F}$ and $\mathcal{C}$ with bandwidth b if for any $F \in \mathcal{F}$, there are:*

- *positive integers $b_1, \dots, b_n \in \mathbb{Z}^{\geq 0}$ so that $\sum_j b_j \leq b$;*
- *functions $g_1, \dots, g_n$ so that $g_j : \mathbb{F} \rightarrow \{0, 1\}^{b_j}$;*
- *and a function $G : \{0, 1\}^{\sum_j b_j} \rightarrow \mathbb{F}$*

so that for all $\mathbf{x} \in \mathbb{F}^k$, if $\mathbf{c} = \mathcal{C}(\mathbf{x})$, then $G(g_1(c_1), g_2(c_2), \dots, g_n(c_n)) = F(\mathbf{x})$. We denote the scheme by $\Phi : F \mapsto (g_1, g_2, \dots, g_n, G)$ that maps $F \in \mathcal{F}$ to the maps g_j and G .

If there is a set $\mathcal{I} \subset [n]$ so that $b_j = 0$ for all $F \in \mathcal{F}$ and for all $j \in \mathcal{I}$, we say that Φ tolerates failures in $\mathcal{I}$.

► **Remark 2** (More general alphabets). More generally, one could define an evaluation scheme for codes $\mathcal{C} : \Sigma_0^k \rightarrow \Sigma_1^n$ for arbitrary input/output alphabets. In this paper, we focus on linear functions and MDS codes, so we state Definition 1 with $\Sigma_0 = \Sigma_1 = \mathbb{F}$ being some finite field.

► **Remark 3** (Knowledge of $\mathcal{I}$). We note that in Definition 1, the set of failed nodes $\mathcal{I}$ tolerated by a scheme is a property of that particular scheme; a stronger definition might demand that the same scheme tolerates *any* set of failed nodes of a particular size. In the distributed storage example above, this weaker definition means that the nodes may need to know which nodes have failed in order to decide which scheme to use. This mirrors the set-up in regenerating codes, discussed below, where the identity of the (single) failed node is assumed to be known.

Notions related to Definition 1 have been studied before, for particular families of functions and/or particular codes. We mention a few of these below, and discuss them more in Section 1.3 (Applications of our results) and Section 1.4 (Related work).

- **Regenerating codes.** In the model of distributed storage described above, there has been a great deal of work on *regenerating codes*, which aim to repair one node failure with low download bandwidth (see, e.g., [10]). This is a special case of Definition 1 when

$\mathcal{F}$ is the family of functions $f_i(\mathbf{x}) = \mathcal{C}(\mathbf{x})_i$, for $i = 1, \dots, n$, and where $\mathcal{I} = \{i\}$. We note that if the code is systematic, this allows us to recover the dictator functions $f_i(\mathbf{x}) = x_i$.

- **Gradient Coding.** The goal of *gradient coding* is to speed up distributed gradient descent in the presence of *stragglers*, that is, compute nodes that may be slow or unresponsive [28]. In this model, the data is $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(k)})$ where each $\mathbf{x}^{(i)} \in \mathbb{R}^d$ for some d . The data $\mathbf{X}$ is distributed using a code among n workers, so that worker i receives $\mathbf{c}^{(i)} \in \mathbb{R}^{d'}$, for a codeword $\mathbf{C} = (\mathbf{c}^{(1)}, \dots, \mathbf{c}^{(n)})$. At each timestep the parameter server (PS) has an iterate $\boldsymbol{\theta} \in \mathbb{R}^d$, which it broadcasts to the workers. Each worker i that has not failed computes a local function $g_i(\mathbf{c}^{(i)})$ and returns it to the PS. The PS then uses these messages to recover the gradient of some loss function, $\nabla \mathcal{L}(\mathbf{X}; \boldsymbol{\theta})$. One goal of gradient coding is to tolerate stragglers in any set $\mathcal{I}$ of some fixed size, while minimizing the communication bandwidth from the workers to the PS (e.g., [29]). This can be cast as a special case of a strengthening of Definition 1 (with different input/output alphabets as per Remark 2 and which can tolerate any small set $\mathcal{I}$ of failed nodes as per Remark 3), where $\mathcal{F}$ is the family of functions given by possible gradients: $\mathcal{F} = \{F_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \mathbb{R}^d\}$, where $F_{\boldsymbol{\theta}}(\mathbf{X}) = \nabla \mathcal{L}(\mathbf{X}; \boldsymbol{\theta})$.
- **Homomorphic Secret Sharing.** In *secret sharing*, a secret s is shared among n parties, so that some coalitions of parties can recover the secret while others learn nothing about it. A classic example is *Shamir's scheme*, which is essentially a Reed-Solomon (RS) code: let $\mathcal{C}$ be a RS code of dimension k and length $n + 1$. To share a secret $s \in \mathbb{F}$, we pick a random codeword $\mathbf{c} \in \mathcal{C}$ such that $c_0 = s$, and we send c_i to party i . Any k parties can recover $\mathbf{c}$ and hence s , but any $k - 1$ parties learn nothing about s . In *single-client Homomorphic Secret Sharing* (HSS) [3, 2, 4], one additionally asks that the parties be able to locally compute messages $g_i(c_i)$ so that a referee can compute a function $f(s)$ of the secret from these messages.¹ The HSS property has applications in Private Information Retrieval and Secure Multiparty Computation (see, e.g., [2]). In some applications, it is desirable that the messages $g_i(c_i)$ be short, in which case the HSS scheme is said to be *compact*. Low-bandwidth function evaluation is related to (information-theoretic, not-necessarily-additive) compact HSS, in the sense that a low-bandwidth function evaluation scheme for a Reed-Solomon code $\mathcal{C}$ and with $\mathcal{F} = \{F : \mathbf{x} \mapsto f(\mathcal{C}(\mathbf{x})_0)\}$ gives a single-client HSS protocol for Shamir's scheme; the bandwidth of the scheme corresponds to the compactness of the messages $g_i(c_i)$. More generally, if $\mathcal{C}$ represents a secret sharing scheme, then a low-bandwidth function evaluation scheme for $\mathcal{C}$ yields a compact single-client HSS protocol for that scheme.

Given the numerous places that notions related to Definition 1 have appeared, we believe it will be fruitful to study Definition 1 in generality. In this paper we begin this general study by considering what is arguably most natural class of functions (after the indicator functions $f(\mathbf{x}) = \mathcal{C}(\mathbf{x})_i$ studied in regenerating codes): the class of linear functions. Our main results, described next in Section 1.2, are low-bandwidth evaluation schemes for Reed-Solomon codes, for classes of linear functions. In Section 1.3, we mention several applications of our results.

1.2 Our Results

Our main results hold for Reed-Solomon codes, defined below.

► **Definition 4** (Reed-Solomon Code). *Let $\mathbb{F}$ be a finite field and suppose $n \leq |\mathbb{F}|$. Let $\alpha_1, \alpha_2, \dots, \alpha_n \in \mathbb{F}$ be distinct evaluation points. The **Reed-Solomon Code (RS)** of dimension k and length n with evaluation points $\alpha_1, \dots, \alpha_n$ is the map $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ given by $\mathcal{C}(\mathbf{f}) = (f(\alpha_1), \dots, f(\alpha_n))$, where for $\mathbf{f} \in \mathbb{F}^k$, we define $f(X) = \sum_{i=0}^{k-1} f_i X^i$.*

¹ Typically, $f(s)$ should be a sum of the messages, in which case the HSS scheme is called *additive*.

Our contributions are as follows.

1. **A framework for computing linear functions on RS-encoded data.** We provide a framework for developing low-bandwidth evaluation schemes for RS codes and for families $\mathcal{F}$ of linear functions over extension fields. This begins with a general linear-algebraic characterization (similar to the characterization of [14] for regenerating codes) that applies to *any* linear code. However, we go beyond that, building on it to develop a framework for RS codes in particular. The linear-algebraic characterization for any linear code is given in Section 3, and the framework for RS codes is given in Section 4.
2. **Low-bandwidth schemes for computing any linear function on RS-encoded data, up to the Singleton bound.** Our main theorem, Theorem 19, can be summarized/simplified as follows:

► **Theorem 5** (Simplified; see Theorem 19). *Let $\varepsilon > \gamma > 0$. There is some $q_0 = \Theta\left(\frac{1}{\varepsilon - \gamma}\right)$ so that the following holds for sufficiently large n and for any prime power $q \geq q_0$. Let $Q = q^t$ for any $t \geq 2$. Let $n = Q$ and let $k = (1 - \varepsilon)n$. Let $\mathcal{C}$ be the RS code over $\mathbb{F} = \mathbb{F}_Q$ of dimension k and length n , with evaluation points all of $\mathbb{F}$. Let $\mathcal{F}$ be the set of all linear functions $F : \mathbb{F}^k \rightarrow \mathbb{F}$. Let $\mathcal{I} \subseteq [n]$ be any set with $|\mathcal{I}| < \gamma n$. Then there is an evaluation scheme for $\mathcal{C}$ and $\mathcal{F}$ that tolerates failures in $\mathcal{I}$, with bandwidth (measured in bits) of $b = O\left(\frac{n \log(q)}{\varepsilon - \gamma}\right)$.*

We make a few remarks about Theorem 5:

- The naive scheme (downloading enough information to recover $\mathbf{x}$, and then computing $F(\mathbf{x})$) requires $k \log(Q) = \frac{n \log n}{1 - \varepsilon}$ bits of bandwidth. Thus, when ε, γ, q are constant, our scheme gives an asymptotic improvement of a factor of $\log n$ over the naive scheme. (Notice that $|\mathbb{F}| = Q = q^t$, so we may choose q to be constant and allow $n = Q$ to grow by growing t).
- Our scheme can tolerate a γ fraction of failures, where γ can be arbitrarily close to ε . Since the rate of the code is $1 - \varepsilon$, by the Singleton bound the relative distance can be at most ε , and so this is optimal.
- One may wonder about lower bounds on the bandwidth. In the full version [26] we show that $b \geq n \log_q \left(\frac{n}{n - k + 1} \right) \approx n \log_q(1/\varepsilon)$ is necessary for Reed-Solomon codes, and that a similar result holds for any MDS code. This shows that the linear dependence on n in Theorem 5 is optimal for constant ε and q , although we leave it as an open question to pin down the correct dependence on ε and γ .

We also give a simpler version of Theorem 19, Theorem 18, which does not tolerate any failures and which only works up to the rate $1/2$. Our main reason for presenting Theorem 18 is for exposition, as the proof is simpler, but we are also able to get more precise constants. In particular, we show that RS codes of rate approaching $1/2$ (for sufficiently large q , where $Q = q^t$ is the size of $\mathbb{F}$) can compute any linear function with bandwidth at most $n \lceil \log q \rceil$. When $q = 2$, we get an RS code of rate $1/4$ with bandwidth n bits, or *only one bit* from each node.

3. **Applications in distributed storage, coded computation, and homomorphic secret sharing.** Our results have applications in several domains. We elaborate on these next in Section 1.3.

1.3 Applications

As noted above, low-bandwidth function evaluation shows up in several settings, and our work has natural applications in these areas. We briefly mention a few potential applications of Theorem 5 to further motivate our results. First, we make two remarks about the generality of our scheme.

► **Remark 6 (Non-linear functions).** Our framework can also be used to efficiently compute certain non-linear functions, for example $\mathbf{x} \mapsto \sum_{i=1}^k x_i^2$. To see this, we first suppose without loss of generality that the Reed-Solomon code has a systematic encoding, so that $\mathbf{x}$ is encoded as $(f(\alpha_1), \dots, f(\alpha_n))$ where $f \in \mathbb{F}[X]$ is the unique polynomial of degree at most $k-1$ so that $f(\alpha_i) = x_i$ for $i = 1, \dots, k$. Then define $g(X) = f(X)^2$ and observe that each node $i = 1, \dots, n$ can locally compute $g(\alpha_i)$. Thus, we can apply our scheme to the Reed-Solomon code of dimension $2k-1$ to recover the linear combination $\sum_{i=1}^k g(\alpha_i) = \sum_{i=1}^k x_i^2$.

► **Remark 7 (Prime fields).** Our approach requires that $\mathbb{F}$ be an extension field over a base field $\mathbb{B}$. However, in many applications (including those discussed below), it is desirable to work over a prime field. The reason is that often we actually want to work over the reals or the integers, and these can be nicely embedded in $\mathbb{F}_p$ for a large enough prime p . Fortunately, for certain linear functions F , our approach can still be used to save bandwidth when we wish to amortize several computations over prime fields.

In more detail, suppose that $\mathbb{B} = \mathbb{F}_p$ for a large prime p , and let $\mathbb{F} = \mathbb{F}_{p^t}$. Let $\zeta_1, \dots, \zeta_t$ be a basis for $\mathbb{F}$ over $\mathbb{B}$. Suppose that the linear function we want to compute is $F(\mathbf{x}) = \mathbf{b}^T \mathbf{x}$, where $\mathbf{b} \in \mathbb{B}^k$ has coefficients in the base field $\mathbb{B}$. This is the case, for example, in Remark 6 when we want to compute the ℓ_2 norm: all of the coefficients are 1. It is also the case when the data $\mathbf{x}$ represents a histogram and we'd like to take the sum of certain buckets: all of the coefficients are 0 or 1. If $\mathbf{b} \in \mathbb{B}^k$, then we can proceed as follows. View the data $\mathbf{x} \in \mathbb{F}^k$ as t data points in $\mathbb{B}^k$. That is, we write $x_j = \sum_{i=1}^t y_{i,j} \zeta_i$, and interpret $\mathbf{x}$ as t vectors $\mathbf{y}^{(i)} = (y_{i,1}, y_{i,2}, \dots, y_{i,k})$. If we use our scheme to compute $F(\mathbf{x})$, then we have computed

$$F(\mathbf{x}) = \sum_{j=1}^k b_j x_j = \sum_{j=1}^k b_j \left(\sum_{i=1}^t \zeta_i y_{i,j} \right) = \sum_{i=1}^t \zeta_i \mathbf{b}^T \mathbf{y}^{(i)} = \sum_{i=1}^t \zeta_i F(\mathbf{y}^{(i)}).$$

Since $F(\mathbf{y}^{(i)}) \in \mathbb{B}$, and since the ζ_i form a basis for $\mathbb{F}$ over $\mathbb{B}$, we can now read off the values $F(\mathbf{y}^{(i)})$. This allows us to compute t evaluations of F on vectors $\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(t)} \in \mathbb{B}^k$, using bandwidth $O(n \log p)$ (assuming that the rate $1 - \varepsilon$ of the code and fraction $\gamma < \varepsilon$ of failed nodes are constants). In contrast, the naive computation would require $O(tn \log p)$ bits. So for such linear functions F , our scheme can do $t = \log_p n$ computations for the bandwidth cost of a single computation in the naive scheme.

Distributed Storage

The application to distributed storage was described in Section 1. In this context, Theorem 5 gives a method to compute any linear function of data stored on a distributed storage system with non-trivial download bandwidth. The reader may be wondering about the *upload* bandwidth: don't we need to communicate the function F to each node? The reason that we focus on the download bandwidth (as is also the case for regenerating codes) is because of the way that files are stored in a typical distributed storage system. In more detail, a large file $\mathbf{x}$ will be broken up into blocks $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)} \in \mathbb{F}^k$, where M is very large, and each $\mathbf{x}^{(i)}$ will be encoded as $\mathbf{c}^{(i)}$, so that the j 'th node stores $\{c_j^{(i)} : i \in [M]\}$. With this set-up, the evaluation scheme of Theorem 5 would be run independently on each of the blocks, so that the upload cost is just the cost of broadcasting F , while the download cost is M times the bandwidth guaranteed in the theorem. Since M is large, the download cost dominates the upload cost, and Theorem 5 yields real bandwidth savings over the naive scheme.

Coded Computation and Low-Bandwidth Matrix-Vector Multiplication

Suppose we would like to distribute some data $\mathbf{X}$ among n worker nodes and perform a computation $f(\mathbf{X})$ in a distributed way. A body of work [17, 11, 30] on *coded computation* has proposed introducing redundancy in the data assignment, with the goal of tolerating stragglers (worker nodes that may be slow or non-responsive): that is, we would like responses from any $\tilde{k}$ out of n workers to determine $f(\mathbf{X})$. There are two lines of work in coded computation. One line of work adds redundancy by replicating and appropriately distributing data (for example the work on gradient coding mentioned above [29], or a line of work aimed at general MapReduce computations [21]), and aims to minimize download bandwidth. Unfortunately, because the coding is done by replication, the rate of the resulting code is necessarily small. A second line of work adds redundancy through true “coding” (eg, taking nontrivial linear combinations). This allows for high-rate codes without much overhead in terms of the total computational load, but instead of focusing on bandwidth, this line of work has focused on minimizing the number $\tilde{k}$ of nodes that need to respond. Several works in this second line have focused on linear functions, like matrix-vector multiplication [17, 11] or Fourier transforms [31]; to the best of our knowledge, none of these have focused on download bandwidth beyond minimizing the number $\tilde{k}$ of workers that need to respond.

Our work provides a way to interpolate between these two lines of work. That is, our work gives coded computation schemes for linear functions that *both* can have low download bandwidth *and* that can use non-replication-based coding to achieve a high rate. In particular, Theorem 5 shows that we can use a rate $1 - \varepsilon$ RS code, with bandwidth that scales like n/ε , saving an $O(\log n)$ factor when ε is constant. As per Remarks 6 and 7 above, this approach can be used effectively to compute, say, ℓ_p norms over the reals, even though our Theorem 5 is stated for linear functions over extension fields.

We note that this is not directly comparable to prior work for coded computation of linear functions (eg, [17, 11, 32]) for two reasons. First, those works have focused on computations with a larger output (eg, matrix-vector multiplication, where the output is a vector rather than a scalar), while our approach is most effective when the desired output is a scalar. Second, in much of the work on coded computation, the identities of the stragglers are not known to the other worker nodes. In our approach, since the scheme may depend on the set $\mathcal{I}$ of failed nodes, the parameter server would have to broadcast this information, which may not be practical. However, we note that the problem is still interesting even if there are no stragglers, simply to reduce download bandwidth (as in [21]); or when the “stragglers” can be planned (for example to do load balancing between multiple tasks).

Homomorphic Secret Sharing

We have described the basic set-up for Homomorphic Secret Sharing (HSS) above. Our scheme immediately gives a compact single-client HSS scheme for linear functions, by sharing a secret $\mathbf{x} \in \mathbb{F}^k$ using a generalization of Shamir’s scheme (as in [13]) as follows. Let $\mathbf{x} \in \mathbb{F}^k$ be a secret. Let $\tilde{k} > k$, so that $\tilde{k} + k < n$. We encode $\mathbf{x}$ with a systematic Reed-Solomon code, so that $x_i = f(\beta_i)$ for $i = 1, \dots, k$, where f is a random polynomial of degree at most $\tilde{k} - 1$ so that this is true, and where $\beta_1, \dots, \beta_k \in \mathbb{F}$ are fixed evaluation points. Then we distribute shares $f(\alpha_1), \dots, f(\alpha_n)$ to the n parties, where $\alpha_i \in \mathbb{F} \setminus \{\beta_1, \dots, \beta_k\}$. Now, any $\tilde{k}$ parties can recover the secret, while any $\tilde{k} - k$ learn nothing about it. Theorem 5 (treating the evaluation points $\beta_1, \dots, \beta_k$ as the unavailable nodes in $\mathcal{I}$) ensures that as long as $\varepsilon = 1 - \tilde{k}/n$ and $\gamma = k/n$ are constants with $\gamma < \varepsilon$, then each party can compute a small local share $g_i(f(\alpha_i))$, which can then be combined to recover a linear function $F(\mathbf{x})$.

As noted in Remarks 6 and 7 above, this approach can also be used for amortizing the computation of certain (possibly nonlinear) functions over prime fields.

1.4 Related Work

First, we mention two works that are similar in flavor to ours in that the aim is to compute functions on data encoded with an error correcting code, although the models are quite different. The first of these is [7], which studies the notion of *error-correcting data structures*. In that work, a vector $\mathbf{x}$, thought of as a database, is encoded as a data structure $\mathcal{C}(\mathbf{x})$; as in our work, the goal is to efficiently compute some function (e.g., perform a membership query) on $\mathbf{x}$ given access to $\mathcal{C}(\mathbf{x})$, possibly in the presence of noise. However, that work differs from ours because (a) they consider query complexity (rather than bandwidth) as the notion of efficiency; and (b) the noise they consider is errors (rather than erasures). Thus, in some sense, the work [7] generalizes locally decodable codes in the same direction that we generalize regenerating codes. The second work that is similar in flavor is the recent work [18] on *function correcting codes*. In that work, a sender Alice sends a message $\mathbf{x}$ over a noisy channel to a receiver Bob who is only interested in some function $f(\mathbf{x})$. The main focus of that work is on the trade-off between the amount of noise in the channel and the rate of the code, given that Bob can recover $f(\mathbf{x})$. This differentiates their problem from ours because they do not study any notion of efficiency (like bandwidth or query complexity) on Bob's end.

As mentioned above, notions related to Definition 1 arise in a variety of contexts, including in regenerating codes, coded computation, and homomorphic secret sharing. We survey related work in these areas below.

Regenerating codes

The body of work most related to ours is that of regenerating codes. Regenerating codes were introduced in [9] and have seen a huge amount of work since then. The work most related to ours is the study of scalar² MDS codes, including RS codes. This was initiated by [25], and further developed in a line of work including [14, 27]. These works give *repair schemes* for RS codes, which can be seen as evaluation schemes for RS codes and for the class of functions $\mathcal{F} = \{F_i : \mathbf{x} \mapsto \mathcal{C}(x)_i : i \in [n]\}$. The work [14] gives a characterization of repair schemes for MDS codes. This characterization inspires our Definition 8 and Proposition 9, which gives a similar formulation for evaluation schemes for linear codes and classes of linear functions. However, our framework for RS codes developed in Section 4 is quite different than the approach in [14]. In more detail, in [14], the goal is to choose dual codewords $\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(t)}$, so that they give rise to low-dimensional $\mathbb{B}$ -subspaces. In contrast, our approach is to go the other way around: we first pick the low-dimensional $\mathbb{B}$ -subspaces, and show how they give rise to appropriate dual codewords.

Coded computation

As mentioned above, there are two main lines of work in coded computation. We refer the reader to [19] for a survey. One line of work has focused on coding for stragglers and has used “true” coding (in the sense that linear combinations of the original data are stored, rather than

² In the regenerating codes literature, a *scalar* MDS code is one that is linear over its alphabet, as opposed to a *vector* MDS code, which is linear over a smaller field.

repeated blocks). In our framework, stragglers correspond to the set $\mathcal{I}$ of failures, the code maps some data $\mathbf{x}$ to a codeword $\mathbf{c}$ that is distributed to workers, and the goal is to compute some function $F(\mathbf{x})$ from computations $g_i(c_i)$ performed by the worker i on their part of the encoded data c_i . (We note that typically in these settings the symbols x_j and c_i are actually vector or matrix-valued, and the code is applied to each coordinate in parallel). This line of work has considered linear functions like matrix-vector multiplication [17, 11] or Fourier transformations [31], as well as non-linear functions like matrix-matrix multiplication [32] and computation of low-degree polynomials [30]. The main focus has been on minimizing the number of workers required to complete their task before the desired function can be computed, as well as on analyzing when and how much this can speed up computation given stochastic models of stragglers. However, to the best of our knowledge, this line of work has not considered the network bandwidth, which is what we consider here.

A second line of work has also focused on coding for stragglers, but has used replication-based coding. That is, the data $\mathbf{x}$ is separated into blocks, and these blocks are distributed to workers with repetition. For example, worker i might receive blocks 1 and 2, and worker j might receive blocks 2 and 3. This approach is especially common in the area of *gradient coding* [28, 16, 23], where the goal is to compute the function $F_{\theta}(\mathbf{x}) = \nabla L(\mathbf{x}; \theta)$ which is the gradient of a loss function at a current iterate θ . In this set-up, again the main goal is to minimize the number of nodes that need to respond before the function can be computed, but some works like [29] have also considered the download bandwidth. Thus, the goal of [29] is similar to ours, but the approach differs because (a) they are using a replication-based code, and in particular the rate must be low; but (b) their scheme does not depend on the identity of the stragglers, which ours does. We note that there are several relaxations of the gradient coding problem, for example when the stragglers are random and/or the gradient only needs to be approximately computed [6, 23, 20]. Again, those works differ from ours because of the replication-based coding and the different model of stragglers.

A final line of work, starting with [21], has focused on minimizing communication bandwidth, as we do here, but in a different setting. That work considers computation in a general MapReduce framework. In that work, the data is distributed before the Map phase, introducing redundancy via replication. Then the data is shuffled before the Reduce phase; the goal is to reduce the amount of communication in the shuffle. Finally, the Reduce phase occurs, and each node needs to compute the function that they are responsible for. This can be viewed as a decentralized version of our setting where each node wants to compute a (different) function. Key differences between that work and ours are that (a) the coding comes via replication, and (b) the goal is to be able to support generic computation in the MapReduce framework, rather than focusing on specific functions.

Homomorphic Secret Sharing

Homomorphic Secret Sharing was introduced in [3] and has been further explored in [4] and the references therein. As noted above, a single-client compact HSS scheme is related to our definition of low-bandwidth function evaluation, where the code is given by the secret-sharing scheme. The work [3] gave a two-party HSS scheme for any deterministic branching program that is cryptographically secure; this scheme has been optimized in [2], and other works [5, 22, 24] have achieved similar results under different cryptographic assumptions. The work [4] has studied the problem more generally, including under information-theoretic security, and provided lower bounds. While the setup of HSS is quite related to our work, most existing work on HSS is in a very different parameter regime. For example, the two-party case studied in [3] corresponds to a code of length $n = 2$. Additionally, since an MDS code

provides an information-theoretically secure secret-sharing scheme, HSS is most related to our work under information-theoretic security. However, most constructions that we are aware of for HSS have focused on cryptographic security. One exception is the recent work [12], which focuses on the download bandwidth of information-theoretic HSS. However, that work focuses on *multi*-client HSS, where the k secrets in $\mathbf{x} \in \mathbb{F}^k$ must be secret-shared independently of each other; in contrast, the application of our work sketched above is for *single*-client HSS, where the k secrets in $\mathbf{x}$ may be shared jointly.

1.5 Organization

In Section 2 we set notation and give a brief overview of our approach. In Section 3 we introduce our framework for linear functions and linear codes. In Section 4 we introduce our framework for RS codes in particular. In Section 5, we instantiate our framework to prove Theorem 19, the more detailed version of Theorem 5 above. Section 6 concludes with some open questions.

2 Notation and Technical Overview

In this section we set some notation and give a quick technical overview of the main ideas in our work.

2.1 Notation

Throughout, we use $[n]$ to denote the set $\{1, 2, \dots, n\}$. We use bold lowercase letters like $\mathbf{x}$ to denote vectors, and bold uppercase letters like $\mathbf{G}$ to denote matrices. For a vector $\mathbf{x}$, we use x_i to denote the i 'th coordinate of x . We use $\mathbf{x}_{[i,j]}$ to denote the vector $(x_i, x_{i+1}, \dots, x_j)$. For a polynomial $f(X) = \sum_i f_i X^i$, we define the *degree set* of f to be

$$\text{degSet}(f(X)) = \{i : f_i \neq 0\}.$$

We always work over a field $\mathbb{F} = \mathbb{F}_Q$, where $Q = q^t$ and we will let $\mathbb{B} = \mathbb{F}_q$ be the subfield of $\mathbb{F}$ of size q . With $Q = q^t$ as above, we will make use of the *field trace* of $\mathbb{F}_Q$ over $\mathbb{F}_q$, defined by

$$\text{tr}(X) = \sum_{i=0}^{t-1} X^{q^i}.$$

We note the following two facts about the field trace:

- The field trace is $\mathbb{F}_q$ -linear and its image is contained in $\mathbb{F}_q$.
- The field $\mathbb{F} = \mathbb{F}_Q$ is a vector space over the subfield $\mathbb{B} = \mathbb{F}_q$. Given a basis $\zeta_1, \dots, \zeta_t$ for $\mathbb{F}$ over $\mathbb{B}$, the traces $\text{tr}(\zeta_1 \alpha), \dots, \text{tr}(\zeta_t \alpha)$ uniquely specify $\alpha \in \mathbb{F}$.

We consider *linear* codes $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$. Such a code can be represented by a full-rank *generator matrix* $\mathbf{G} \in \mathbb{F}^{n \times k}$, so that $\mathcal{C}(\mathbf{x}) = \mathbf{G}\mathbf{x}$ for $\mathbf{x} \in \mathbb{F}^k$.

We consider both $\mathbb{F}$ -subspaces of $\mathbb{F}^n$ and $\mathbb{B}$ -subspaces of $\mathbb{F}$ or $\mathbb{F}^n$. To that end, we use span , dim , and $\cdot^\perp$ (with no decoration) to refer to the span, dimension, and orthogonal complement over $\mathbb{F}$. We use $\text{span}_{\mathbb{B}}$, $\text{dim}_{\mathbb{B}}$ and $\cdot^{\perp_{\mathbb{B}}}$ (decorated with a “ $\mathbb{B}$ ”) to denote the span, dimension, and orthogonal complement over $\mathbb{B}$. We define the orthogonal complement over $\mathbb{B}$ as follows. For a $\mathbb{B}$ -vector space $V \subseteq \mathbb{F}$, we define $V^{\perp_{\mathbb{B}}} := \{x \in \mathbb{F} : \text{tr}(xv) = 0 \forall v \in V\}$. For a $\mathbb{B}$ -vector space $\mathcal{V} \subset \mathbb{F}^n$, we define

$$\mathcal{V}^{\perp_{\mathbb{B}}} := \{\mathbf{x} \in \mathbb{F}^n : \text{tr}(\mathbf{x}^T \mathbf{v}) = 0 \forall \mathbf{v} \in \mathcal{V}\}.$$

2.2 Technical Overview

Our approach begins with a general linear-algebraic framework, similar to that from [14] for regenerating codes. Let $\mathbb{B}$ be a subfield of $\mathbb{F}$, and let $\zeta_1, \dots, \zeta_t$ be a basis for $\mathbb{F}$ over $\mathbb{B}$. For a code $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$, let $C = \mathcal{C}(\mathbb{F}^k)$, so C is a subspace of dimension k in $\mathbb{F}^n$ that consists of all codewords. We can associate an evaluation scheme with a sequence of $\mathbb{B}$ -subspaces $V_1, \dots, V_n \subset \mathbb{F}$, by demanding that node j return enough information to evaluate $\text{tr}(c_j \nu)$ for all $\nu \in V_j$. Since V_j is $\mathbb{B}$ -linear, it suffices to send b_j symbols from $\mathbb{B}$, where $b_j = \dim_{\mathbb{B}}(V_j)$. When is this enough information to recover a linear function $F_{\mathbf{p}}(\mathbf{x}) = \mathbf{p}^T \mathbf{x}$?

In Definition 8, we define a *linear evaluation scheme* as a sequence of $\mathbb{B}$ -subspaces $V_1, \dots, V_n \subset \mathbb{F}$ that has a nice relationship to $\mathcal{C}$, and then we show that this nice relationship allows us to recover linear functions $F_{\mathbf{p}}(\mathbf{x})$. More precisely, let

$$\mathcal{W} = V_1^{\perp_{\mathbb{B}}} \times \dots \times V_n^{\perp_{\mathbb{B}}}.$$

We show in Section 3 that if $\text{span}_{\mathbb{F}}(C \cap \mathcal{W})$ has low dimension over $\mathbb{F}$, then there are many linear functions $F_{\mathbf{p}}$ that can be recovered by the scheme derived from the subspaces $V_1, \dots, V_n$. Thus, the goal becomes to find $\mathbb{B}$ -subspaces $V_1, \dots, V_n \subset \mathbb{F}$ so that $\text{span}_{\mathbb{F}}(C \cap \mathcal{W})$ is low-dimensional over $\mathbb{F}$. (Additionally, we need to keep track of *which* linear functions we can recover, but we will gloss over that in this overview). Notice that $\mathcal{W}$ is a $\mathbb{B}$ -vector space, but *not* an $\mathbb{F}$ -vector space. Thus, it is not obvious how to get a handle on the dimension of this span.

In order to control the dimension of $\text{span}_{\mathbb{F}}(C \cap \mathcal{W})$, we specialize to Reed-Solomon codes (rather than any linear code); this is where our analysis departs in similarity from [14]. We do this in Sections 4 and 5. Suppose that we choose $V_i = \text{span}_{\mathbb{B}}(v(\alpha_i))$, where $v(X) \in \mathbb{F}[X]$ is some polynomial. Then our goal becomes to show that

$$\{g \in \mathbb{F}[X] : \deg(g) < k, \text{tr}(g(\alpha_j)v(\alpha_j)) = 0 \quad \forall j \in [n]\} \quad (1)$$

lies in a low-dimensional $\mathbb{F}$ -vector space. Again, this is tricky because “ $\text{tr}(g(\alpha_j)v(\alpha_j)) = 0$ ” is a $\mathbb{B}$ -linear constraint, and we want $\mathbb{F}$ -linear constraints. We turn these $\mathbb{B}$ -linear constraints into $\mathbb{F}$ -linear constraints as follows. Consider the unique polynomial $R(X)$ of degree at most $n - 1$ so that

$$R(X) \equiv \text{tr}(g(X)v(X)) \pmod{p_A(X)},$$

where $p_A(X) = \prod_{j=1}^n (X - \alpha_j)$. Now, if $\text{tr}(g(\alpha_j)v(\alpha_j)) = 0$ for all j , then R vanishes everywhere and is thus identically zero. The polynomial R is a bit tricky to write down, but if the evaluation points are all of $\mathbb{F}$, then $p_A(X) = X^Q - X$, and in fact taking the residue of $R(X)$ modulo $p_A(X)$ is tractable. Thus, our strategy is to expand out $R(X)$ and choose the coefficients of v carefully so that the coefficient on some term X^d is of the form $\sum_{\ell} v_{d-\ell} g_{\ell}$. Since that coefficient must be zero – because $R(X)$ is identically zero – this gives us an $\mathbb{F}$ -linear constraint on the polynomial g . If we get enough linearly independent $\mathbb{F}$ -linear constraints this way, we can show that the space (1) lies in a low-dimension $\mathbb{F}$ -vector space, which in turn will show that there are many $F_{\mathbf{p}}$ that can be recovered by the scheme associated with $V_i = \text{span}_{\mathbb{B}}(v(\alpha_i))$. (Again, in this overview we gloss over the fact that we actually want to know *which* functions $F_{\mathbf{p}}$ can be recovered this way: by keeping track of exactly which linear constraints we get, we are able to design the polynomial $v(X)$ so that we can control this.)

The approach above is sufficient to design a scheme for codes of rate up to $1/2$, that doesn’t tolerate any failures $\mathcal{I}$. As a warm-up, we present this result as Theorem 18. In order to extend our result to get Theorem 19, the more detailed version of Theorem 5 above,

we must choose several polynomials $v^{(1)}, \dots, v^{(s)}$, increasing the bandwidth by a factor of s . There are two main ideas here. First, in order to make the rate larger than $1/2$ in the scheme from Theorem 18, we must restrict not only the coefficients of v but also the coefficients of g . This results in a scheme for a subset $\mathcal{F}'$ of linear functions. By repeating this several times, we are able to recover all of the linear functions. Second, in order to handle failures in an arbitrary set $\mathcal{I}$, we choose the polynomials $v^{(r)}(X)$ to additionally vanish on the set $\mathcal{I}$. Indeed, since the subspace V_j given by a polynomial $v(X)$ is $V_j = \text{span}_{\mathbb{B}}(v(\alpha_j))$, if v vanishes on $\mathcal{I}$ then $V_j = \{0\}$ for all $j \in \mathcal{I}$. Thus, the dimension is zero, and the j 'th node does not need to return any information.

We give the framework for general linear codes, and explain why $\text{span}_{\mathbb{F}}(C \cap \mathcal{W})$ is important, in Section 3. We develop our “pick nice polynomials $v(X)$ ” framework in Section 4. Finally, we instantiate our framework for a full-length RS code and analyze it in Section 5.

3 Framework for linear functions and any linear code

For the rest of the paper, we focus on the special case where $\mathcal{C}$ is a linear code, and where $\mathcal{F}$ is a set of linear functions. In this case, evaluation schemes for $\mathcal{F}$ and $\mathcal{C}$ can arise from a simple linear-algebraic condition, defined next.

► **Definition 8** (Linear Evaluation Schemes). *Let $\mathbb{B}$ be a subfield of $\mathbb{F}$, and let $\zeta_1, \dots, \zeta_t \in \mathbb{F}$ be a basis for $\mathbb{F}$ over $\mathbb{B}$. Let $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ be a linear code, and let $\mathbf{G} \in \mathbb{F}^{n \times k}$ be a generator matrix for $\mathcal{C}$. Let $\mathbf{p} \in \mathbb{F}^k$ and let $\mathbf{w} \in \mathbb{F}^n$ be any vector so that $\mathbf{p} = \mathbf{G}^T \mathbf{w}$. (Note that such a vector exists since $\mathbf{G}$ has full column-rank).*

Suppose that $V_1, \dots, V_n \subset \mathbb{F}$ are $\mathbb{B}$ -subspaces, so that $\dim_{\mathbb{B}}(V_j) = b_j$. Let $\mathcal{V} = V_1 \times \dots \times V_n \subset \mathbb{F}^n$. We say that $(V_1, \dots, V_n)$ provide a linear evaluation scheme for $\mathbf{p}$ and $\mathcal{C}$ (with respect to $\{\zeta_1, \dots, \zeta_t\}$) if for all $i \in [t]$,

$$\zeta_i \mathbf{w} \in \mathcal{C}(\mathbb{F}^k)^{\perp} + \mathcal{V}$$

The bandwidth of the scheme is $(\sum_{j=1}^n b_j) \lceil \log |\mathbb{B}| \rceil$. Further, we say that $(V_1, \dots, V_n)$ tolerates failures in $\mathcal{I}$ where $\mathcal{I} := \{j \in [n] : V_j = \{0\}\}$.

For $\mathcal{P} \subseteq \mathbb{F}^k$, we say that a map $\varphi : \mathcal{P} \rightarrow (2^{\mathbb{F}})^n$ provides a linear evaluation scheme for $\mathcal{P}$ and $\mathcal{C}$ if $\varphi(\mathbf{p}) = (V_1, \dots, V_n)$ provides a linear evaluation scheme for $\mathbf{p}$ and $\mathcal{C}$ for all $\mathbf{p} \in \mathcal{P}$. For a set $\mathcal{I} \subset [n]$, we say that φ tolerates failures in $\mathcal{I}$ if, for all $\mathbf{p} \in \mathcal{P}$, $\varphi(\mathbf{p})$ tolerates failures in $\mathcal{I}$.

The following proposition explains why a linear evaluation scheme φ indeed gives us an evaluation scheme for a set of linear functions.

► **Proposition 9.** *Suppose that φ provides a linear evaluation scheme for $\mathcal{P}$ and $\mathcal{C}$, with bandwidth b . Then there is an evaluation scheme for the class of functions*

$$\mathcal{F} = \{F_{\mathbf{p}} : \mathbf{x} \mapsto \mathbf{x}^T \mathbf{p} : \mathbf{p} \in \mathcal{P}\}$$

and $\mathcal{C}$ with bandwidth b . Moreover, for $\mathcal{I} \subset [n]$, this evaluation scheme tolerates failures in $\mathcal{I}$ if φ does.

The proof of Proposition 9 is similar to the approach of [14]. We omit the proof in this extended abstract and refer the reader to the full version of the paper [26]. In the next lemma, we reformulate the condition in Definition 8 in a way that will be helpful going forward. Again, we omit the proof, which can be found in the full version.

► **Lemma 10.** Let $\mathcal{V} = V_1 \times V_2 \times \cdots \times V_n \subset \mathbb{F}^n$, where each V_i is a $\mathbb{B}$ -subspace of $\mathbb{F}$. Let $\mathcal{W} = W_1 \times W_2 \times \cdots \times W_n \subset \mathbb{F}^n$, where $W_i = V_i^{\perp_{\mathbb{B}}}$. Let $\mathcal{C} : \mathbb{F}^k \rightarrow \mathbb{F}^n$ be a linear code. Let $\zeta_1, \dots, \zeta_t$ be a basis for $\mathbb{F}$ over $\mathbb{B}$. Then for any $\mathbf{w} \in \mathbb{F}^n$,

$$\zeta_i \mathbf{w} \in \mathcal{C}(\mathbb{F}^k)^{\perp} + \mathcal{V} \quad \forall i \in [t] \quad \text{if and only if} \quad \mathbf{w} \in (\text{span}_{\mathbb{F}}(\mathcal{C}(\mathbb{F}^k) \cap \mathcal{W}))^{\perp}.$$

4 Framework for linear functions and RS codes

The framework in Section 3 was valid for any linear code $\mathcal{C}$. Now, we specialize to Reed-Solomon codes in order to leverage this characterization. We begin with a few definitions that will be useful for our framework.

► **Definition 11.** Let $A = (\alpha_1, \dots, \alpha_n)$, so that $\alpha_i \in \mathbb{F}$ are distinct. Define

$$p_A(X) = \prod_{j=1}^n (X - \alpha_j).$$

For a non-negative integer j and for $i \in \{0, 1, \dots, t-1\}$, define $\sigma_i(j) \subset \mathbb{Z}$ to be

$$\sigma_i(j) = \text{degSet} \left(\overline{X^{jq^i}} \right),$$

where $\overline{X^{jq^i}}$ is the unique polynomial of degree at most $n-1$ so that

$$\overline{X^{jq^i}} \equiv X^{jq^i} \pmod{p_A(X)}.$$

We note that σ_i depends on the choice of A , but we suppress this dependence in the notation for readability.

► **Remark 12.** For any $j < n$, we have $\sigma_0(j) = \text{degSet} \left(\overline{X^j} \right) = \text{degSet} (X^j) = \{j\}$.

► **Remark 13.** While for general A , σ_i may be quite complicated, for some sets A it is relatively simple. For example, if $A = \mathbb{F}$, then $p_A(X) = X^Q - X$, and $\sigma_i(j) = \{jq^i \pmod{Q-1}\}$, where

$$x \pmod{Q-1} := \begin{cases} y \in \{1, \dots, Q-1\} \text{ so that } y \equiv x \pmod{Q-1} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}.$$

In particular, if we write $x \in \{0, 1, \dots, Q-1\}$ in base- q as $x = \sum_{b=0}^{t-1} x_b q^b$ for $x_b \in \{0, \dots, q-1\}$, then

$$\sigma_i(x) = \left\{ \sum_{b=0}^{t-1} x_b q^{b+i} \pmod{t-1} \right\}$$

is a circular shift of this expansion.

► **Definition 14.** Let $0 < k \leq n$ and consider the Reed-Solomon code $\mathcal{C}$ of dimension k with evaluation points $A = (\alpha_1, \dots, \alpha_n)$ over $\mathbb{F}$. Let $j_{\min}, j_{\max}, d$ be positive integers so that $j_{\min} \leq j_{\max}$ and $j_{\min} < d$. We say that $(j_{\min}, j_{\max}, d)$ is good for $\mathcal{C}$ if all of the following hold:

1. $d < n$ and $j_{\max} + k - 1 < n$;
2. for all $i \in \{1, \dots, t-1\}$, $d \notin \bigcup_{j=j_{\min}}^{j_{\max}+k-1} \sigma_i(j)$; and
3. $d \in \bigcup_{j=j_{\min}}^{j_{\max}+k-1} \sigma_0(j)$,

where above σ_i is defined as in Definition 11 with respect to A . Given some $d, j_{\min}, j_{\max}$, we define $\ell_{\min}$ and $\ell_{\max}$ by

$$\ell_{\min} = \max\{0, d - j_{\max}\} \quad \text{and} \quad \ell_{\max} = \min\{k - 1, d - j_{\min}\}. \quad (2)$$

► **Definition 15.** Fix $(d, j_{\min}, j_{\max})$, and let $\ell_{\min}, \ell_{\max}$ be as in (2). Let $\mathbf{p} \in \mathbb{F}^k$, so that $\text{Supp}(\mathbf{p}) \subseteq [\ell_{\min}, \ell_{\max}]$. We say that a polynomial

$$v(X) = \sum_{j=j_{\min}}^{j_{\max}} v_j X^j$$

is consistent with $\mathbf{p}$ (with respect to $(d, j_{\min}, j_{\max})$), if $v_j = p_{d-j}$ whenever $d - j \in [0, k - 1]$.

Notice that, for any $\mathbf{p}$ as in Definition 15, there is some polynomial $v(X)$ consistent with $\mathbf{p}$, given by $v(X) = \sum_{j=\max\{d-k+1, j_{\min}\}}^{\min\{d, j_{\max}\}} p_{d-j} X^j$.

With these definitions, we have the following lemma.

► **Lemma 16.** Let $0 < k \leq n$ and consider the Reed-Solomon code $\mathcal{C}$ of dimension k with evaluation points $A = (\alpha_1, \dots, \alpha_n)$ over $\mathbb{F}$. Suppose that $(j_{\min}, j_{\max}, d)$ is good for $\mathcal{C}$, and let $\ell_{\min}, \ell_{\max}$ be as in Definition 14. Then for all $\mathbf{p} \in \mathbb{F}^k$ so that $\text{Supp}(\mathbf{p}) \subseteq [\ell_{\min}, \ell_{\max}]$, and for all $v(X)$ consistent with $\mathbf{p}$, there exist $\mathbb{B}$ -subspaces $V_1, \dots, V_n \subset \mathbb{F}$ with $\dim_{\mathbb{B}}(V_j) \leq 1$ for all j , so that the following holds:

Let $\mathbf{g} \in \mathbb{F}^k$ and let $g(X) = \sum_{\ell=0}^{k-1} g_{\ell} X^{\ell}$. Suppose that $g(\alpha_j) \in V_j^{\perp_{\mathbb{B}}}$ for all $j \in [n]$. Then $\mathbf{g}^T \mathbf{p} = 0$.

Further, $V_j = \{0\}$ for all $j \in \mathcal{I}$, where $\mathcal{I} = \{j \in [n] : v(\alpha_j) = 0\}$. (Notice that $\mathcal{I}$ depends on both $\mathbf{p}$ and the choice of $v(X)$).

Before we prove Lemma 16, we show how to use it to obtain a linear evaluation scheme for $\mathcal{C}$. The following Theorem is our main framework theorem for RS codes.

► **Theorem 17 (Main Framework Theorem).** Let $\mathcal{C}$ be a Reed-Solomon code. Suppose that $(j_{\min}, j_{\max}, d)$ is good for $\mathcal{C}$. Let $\ell_{\min}, \ell_{\max}$ be as in (2), and let

$$\mathcal{P} \subseteq \{\mathbf{p} \in \mathbb{F}^k : \text{Supp}(\mathbf{p}) \subseteq [\ell_{\min}, \ell_{\max}]\}.$$

Then there is a linear evaluation scheme φ for $\mathcal{P}$ and $\mathcal{C}$ with bandwidth at most $n \lceil \log |\mathbb{B}| \rceil$.

Further, for any collection $\{v_{\mathbf{p}}(X) : \mathbf{p} \in \mathcal{P}\}$ so that for each $\mathbf{p} \in \mathcal{P}$, $v_{\mathbf{p}}(X)$ is consistent with $\mathbf{p}$, there exists a scheme φ that tolerates failures in

$$\mathcal{I} := \{j \in [n] : v_{\mathbf{p}}(\alpha_j) = 0 \quad \forall \mathbf{p} \in \mathcal{P}\},$$

with bandwidth is at most $(n - |\mathcal{I}|) \lceil \log |\mathbb{B}| \rceil$.

Proof of Theorem 17, assuming Lemma 16. We prove the “Further” statement, since it implies that first statement. (Indeed, we may take $v_{\mathbf{p}}(X)$ to be any polynomial consistent with $\mathbf{p}$).

Suppose that $(j_{\min}, j_{\max}, d)$ is good for $\mathcal{C}$. Let $\mathbf{p} \in \mathcal{P}$, and suppose that $v_{\mathbf{p}}(X)$ is consistent with $\mathbf{p}$. Let $\mathbf{G} \in \mathbb{F}^{n \times k}$ be a generator matrix for $\mathcal{C}$ and let $\mathbf{w}$ be such that $\mathbf{p} = \mathbf{G}^T \mathbf{w}$, as in Definition 8. Let $V_1, \dots, V_n$ be the $\mathbb{B}$ -subspaces guaranteed for $\mathbf{p}$ and $v_{\mathbf{p}}(X)$ by Lemma 16, and define $W_j = V_j^{\perp_{\mathbb{B}}}$ for $j = 1, \dots, n$. Let $\mathcal{W} = W_1 \times \dots \times W_n$ and let $\mathcal{V} = V_1 \times \dots \times V_n$.

The guarantee of Lemma 16 implies that $\mathbf{c}^T \mathbf{w} = 0$ for all $\mathbf{c} \in \mathcal{C}(\mathbb{F}^k) \cap \mathcal{W}$, so $\mathbf{w} \in (\mathcal{C}(\mathbb{F}^k) \cap \mathcal{W})^{\perp}$. By Lemma 10, this implies that

$$\zeta_i \mathbf{w} \in \mathcal{C}(\mathbb{F}^k)^{\perp} + \mathcal{V} \quad \forall i \in [t],$$

117:14 Low-Bandwidth Recovery of Linear Functions of Reed-Solomon-Encoded Data

where $\zeta_1, \dots, \zeta_t$ is a basis for $\mathbb{F}$ over $\mathbb{B}$. Therefore from Definition 8, $(V_1, \dots, V_n)$ is a linear scheme for $\mathbf{p}$ and $\mathcal{C}$, and the map φ that maps $\mathbf{p}$ to $(V_1, \dots, V_n)$ as above is a linear scheme for $\mathcal{P}$ and $\mathcal{C}$.

Further, Lemma 16 implies that for all $\mathbf{p} \in \mathcal{P}$, if $\varphi(\mathbf{p}) = (V_1, \dots, V_n)$ then $V_j = \{0\}$ for all $j \in \mathcal{I}$. Thus, φ tolerates failures in $\mathcal{I}$.

Finally, we observe that the bandwidth of the scheme is $\log |\mathbb{B}|$ times the number of V_j so that $V_j \neq \{0\}$, which is at most $(n - |\mathcal{I}|) \log |\mathbb{B}|$. $\blacktriangleleft$

Finally, we prove Lemma 16.

Proof of Lemma 16. Suppose that $\mathcal{C}$ is an RS code as in the statement of the lemma, so we have evaluation points $A = (\alpha_1, \dots, \alpha_n)$. Suppose that $(j_{\min}, j_{\max}, d)$ is good for $\mathcal{C}$. Choose $\mathbf{p} \in \mathbb{F}^k$ such that $\text{Supp}(\mathbf{p}) \in [\ell_{\min}, \ell_{\max}]$, and suppose that $v(X) = \sum_{j=j_{\min}}^{j_{\max}} v_j X^j$ is consistent with $\mathbf{p}$. Define $V_j := \text{span}_{\mathbb{B}}(v(\alpha_j))$ for $j \in [n]$. Notice that $\dim_{\mathbb{B}}(V_j) \leq 1$, as desired, and further that $V_j = \{0\}$ if $v(\alpha_j) = 0$. Now suppose that $\mathbf{g} \in \mathbb{F}^k$ so that $g(X) = \sum_{\ell=0}^{k-1} g_\ell X^\ell$ has $g(\alpha_j) \in V_j^{\perp_{\mathbb{B}}}$. We wish to show that $\mathbf{g}^T \mathbf{p} = 0$.

From the definition of V_j and the assumption that $g(\alpha_j) \in V_j^{\perp_{\mathbb{B}}}$ for all $j \in [n]$, we have

$$\text{tr}(v(\alpha_j)g(\alpha_j)) = 0$$

for all j . Consider the unique polynomial $R(X)$ of degree at most $n - 1$ so that

$$R(X) \equiv \text{tr}(v(X)g(X)) \pmod{p_A(X)},$$

where $p_A(X)$ is as in Definition 11. Thus, $R(\alpha_i) = 0$ for all $i \in [n]$. Since $\deg(R) \leq n - 1$, this implies that $R(X) \equiv 0$ is identically zero. Consider the coefficient of X^d in $R(X)$. On the one hand, this is zero. On the other hand, we can compute

$$\begin{aligned} R(X) &= \sum_{i=0}^{t-1} \left(\sum_{j=j_{\min}}^{j_{\max}} v_j X^j \right)^{q^i} \left(\sum_{\ell=0}^{k-1} g_\ell X^\ell \right)^{q^i} \\ &= \sum_{i=0}^{t-1} \sum_{j, \ell} v_j^{q^i} g_\ell^{q^i} X^{q^i(\ell+j)} \end{aligned}$$

Thus, we have

$$0 = (\text{coefficient of } X^d \text{ in } R(X)) = \sum_{i=0}^{t-1} \sum_{j, \ell: d \in \sigma_i(\ell+j)} c_{\ell+j, d, i} v_j^{q^i} g_\ell^{q^i}, \quad (3)$$

where $c_{r, d, i} \in \mathbb{F}$ are the coefficients that arise when we write

$$\overline{X^{rq^i}} = \sum_{d \in \sigma_i(r)} c_{r, d, i} X^d.$$

(Above, as in Definition 11, $\overline{X^{rq^i}}$ refers to the residue modulo $p_A(X)$). Since $(d, j_{\min}, j_{\max})$ is good, Item 2 of Definition 14 says that for all $i \neq 0$, and for all $r \in [j_{\min}, j_{\max} + k - 1]$, $d \notin \sigma_i(r)$. Since $\ell + j \in [j_{\min}, j_{\max} + k - 1]$, this implies that the inner sum on the right

hand side of (3) is empty if $i \neq 0$. Therefore, we have

$$\begin{aligned} 0 &= (\text{coefficient of } X^d \text{ in } R(X)) \\ &= \sum_{j, \ell: d \in \sigma_0(\ell+j)} c_{\ell+j, d, 0} v_j g_\ell \\ &= \sum_{j=j_{\min}}^{j_{\max}} \sum_{\ell=0}^{k-1} \mathbf{1}_{\ell+j=d} v_j g_\ell \end{aligned} \quad (4)$$

$$= \sum_{j=\max\{d-k+1, j_{\min}\}}^{\min\{d, j_{\max}\}} \sum_{\ell=0}^{k-1} \mathbf{1}_{\ell+j=d} v_j g_\ell \quad (5)$$

$$= \sum_{\ell=\ell_{\min}}^{\ell_{\max}} p_\ell g_\ell \quad (6)$$

$$= \sum_{\ell=0}^{k-1} p_\ell g_\ell. \quad (7)$$

Above, we have used in (4) the fact that $\sigma_0(\ell+j) = \{\ell+j\}$ (as per Remark 12, using the assumption that $\ell+j \leq k-1+j_{\max} < n$ as per Definition 14); and the fact that $c_{d,d,0} = 1$ since we have $\overline{X^d} = X^d$ (using the assumption that $d < n$). In (5), we have used the fact that for $j \in [j_{\min}, d-k] \cup [d+1, j_{\max}]$, $\mathbf{1}_{\ell+j=d} = 0$. In (6), we have used the definition (2) of $\ell_{\min}$ and $\ell_{\max}$. And in (7), we have used the fact that $\text{Supp}(\mathbf{p}) \subseteq [\ell_{\min}, \ell_{\max}]$.

This shows that $\mathbf{p}^T \mathbf{g} = 0$, which completes the proof. $\blacktriangleleft$

5 Proof of main theorem

We begin with a warm-up that already gives good schemes for RS codes of rates approaching $1/2$.

► **Theorem 18.** *Let $Q = q^t$, for some $t \geq 2$ and some prime power q . Suppose that $k \leq Q \left(\frac{1}{q} \left\lfloor \frac{q}{2} \right\rfloor \left(1 - \frac{1}{q} \right) \right)$. Let $\mathcal{C}$ be the Reed-Solomon code of dimension k and length $n = Q$ over $\mathbb{F} = \mathbb{F}_Q$. Let $\mathcal{F}$ be the class of all linear functions from $\mathbb{F}^k$ to $\mathbb{F}$:*

$$\mathcal{F} = \{ (F_{\mathbf{y}} : \mathbf{x} \mapsto \mathbf{x}^T \mathbf{y}) : \mathbf{y} \in \mathbb{F}^k \}.$$

Then there is an evaluation scheme for $\mathcal{F}$ and $\mathcal{C}$ with bandwidth $n \lceil \log_2 q \rceil$.

Notice that the rate of the RS code $\mathcal{C}$ in Theorem 18 can be as large as

$$\frac{k}{n} = \frac{1}{q} \left\lfloor \frac{q}{2} \right\rfloor \left(1 - \frac{1}{q} \right) \geq \frac{1}{2} - \frac{3}{2q},$$

which approaches $1/2$ as q grows. We note that for $q = 2$, the rate of $\mathcal{C}$ is $1/4$.

Proof of Theorem 18. We will use Theorem 17 to show that there is a linear scheme for $\mathcal{P} = \mathbb{F}^k$. Then Proposition 9 will imply the theorem.

Choose

$$\begin{aligned} d &= \left\lfloor \frac{q}{2} \right\rfloor q^{t-1} \\ j_{\min} &= \left\lfloor \frac{q}{2} \right\rfloor q^{t-2} + 1 \\ j_{\max} &= Q - k. \end{aligned}$$

We claim that $(d, j_{\min}, j_{\max})$ is good for $\mathcal{C}$. We check the three items in Definition 14:

117:16 Low-Bandwidth Recovery of Linear Functions of Reed-Solomon-Encoded Data

1. Since $n = Q = q^t$, and using the choice of d above, we have $d < n$. We also have $j_{\max} + k - 1 = Q - 1 < n$.
2. As per Remark 13, for this full-length RS code we have $\sigma_i(j) = \{jq^i \bmod^* Q - 1\}$. Thus, the second item in Definition 14 is equivalent³ to showing that for all $i = 1, \dots, t - 1$,

$$\begin{aligned} d &\neq q^i j \bmod^* Q - 1 && \forall j \in [j_{\min}, j_{\max} + k - 1] \\ dq^t \bmod^* Q - 1 &\neq q^i j \bmod^* Q - 1 && \forall j \in [j_{\min}, j_{\max} + k - 1] \\ dq^{t-i} \bmod^* Q - 1 &\neq j \bmod^* Q - 1 && \forall j \in [j_{\min}, j_{\max} + k - 1] \\ dq^{t-i} \bmod^* Q - 1 &\notin [j_{\min}, j_{\max} + k - 1]. \end{aligned}$$

Plugging in the definitions of d , $j_{\min}$ and $j_{\max}$, this is the same as showing that for all $i = 1, \dots, t - 1$,

$$\left\lfloor \frac{q}{2} \right\rfloor q^{t-i-1} \notin \left[\left\lfloor \frac{q}{2} \right\rfloor q^{t-2} + 1, Q - 1 \right].$$

This is true, because for all $i = 1, \dots, t - 1$, we have

$$0 < \left\lfloor \frac{q}{2} \right\rfloor q^{t-i-1} < \left\lfloor \frac{q}{2} \right\rfloor q^{t-2} + 1.$$

3. Finally, using the fact that $\sigma_0(j) = \{j\}$ for all $j \in [j_{\min}, j_{\max} + k - 1]$, the third item is equivalent to showing that $d \in [j_{\min}, j_{\max} + k - 1]$, or that

$$\left\lfloor \frac{q}{2} \right\rfloor q^{t-1} \in \left[\left\lfloor \frac{q}{2} \right\rfloor q^{t-2} + 1, Q - 1 \right],$$

which is true.

Thus, $(d, j_{\min}, j_{\max})$ is good for $\mathcal{C}$. Now we compute $\ell_{\min}, \ell_{\max}$ as in Theorem 17. We have

$$\begin{aligned} \ell_{\min} &= \max\{0, d - j_{\max}\} \\ &= \max\left\{0, \left\lfloor \frac{q}{2} \right\rfloor q^{t-1} - q^t + k\right\} \\ &= 0, \end{aligned}$$

using the fact that $Q = q^t$ and

$$k \leq Q \left(\frac{1}{q} \left\lfloor \frac{q}{2} \right\rfloor (1 - 1/q) \right) \leq Q \frac{1}{q} \left\lfloor \frac{q}{2} \right\rfloor \leq Q \left(1 - \frac{1}{q} \left\lfloor \frac{q}{2} \right\rfloor \right).$$

We also have

$$\begin{aligned} \ell_{\max} &= \min\{k - 1, d - j_{\min}\} \\ &= \min\left\{k - 1, \left\lfloor \frac{q}{2} \right\rfloor q^{t-1} - \left\lfloor \frac{q}{2} \right\rfloor q^{t-2} - 1\right\} \\ &= k - 1, \end{aligned}$$

using the fact that $k \leq Q \left(\frac{1}{q} \left\lfloor \frac{q}{2} \right\rfloor (1 - 1/q) \right)$. Therefore we have

$$\{\mathbf{p} \in \mathbb{F}^k : \text{Supp}(\mathbf{p}) \subseteq [\ell_{\min}, \ell_{\max}]\} = \mathbb{F}^k.$$

By Theorem 17 and the fact that $(d, j_{\min}, j_{\max})$ is good for $\mathcal{C}$, we conclude that there is a linear evaluation scheme φ for $\mathcal{P} = \mathbb{F}^k$, and $\mathcal{C}$, with bandwidth $n \lceil \log q \rceil$, which is what we wanted to show. $\blacktriangleleft$

³ Note here that for all $i \geq 0$, q^i is a unit of $\mathbb{Z}/(Q-1)\mathbb{Z}$, and in particular $q^t \bmod^* Q - 1 = Q \bmod^* Q - 1 = 1$.

The reason that Theorem 18 has rate limited by $1/2$ is that if we were to take k to be larger, the interval $[\ell_{\min}, \ell_{\max}]$ would not be all of $[0, k-1]$. In the full version [26], we show how to modify the construction in Theorem 18 to give a constant number of schemes like the one in Theorem 18, each of which covers a small interval, but which together cover all of $[0, k-1]$. Thus, we can increase the rate of the code to approach 1, at the cost of increasing the bandwidth by a constant factor. While we are at it, we give ourselves enough freedom in order to choose the schemes so that they can tolerate failures in any set $\mathcal{I}$ that is not too large. This results in the following theorem, whose proof is deferred to the full version.

► **Theorem 19.** *Let $Q = q^t$, for some $t \geq 2$. Let $\varepsilon, \gamma > 0$. Let $\delta \geq \gamma + \frac{1}{q}$, and suppose that $\varepsilon > \delta$ and that $(\varepsilon - \delta)q$ is an integer. Suppose that $k \leq Q(1 - \varepsilon)$, and let $\mathcal{C}$ be the Reed-Solomon code of dimension k and length $n = Q$ over $\mathbb{F} = \mathbb{F}_Q$. Let $\mathcal{F}$ be the class of all linear functions from $\mathbb{F}^k$ to $\mathbb{F}$:*

$$\mathcal{F} = \{ (f : \mathbf{x} \mapsto \mathbf{x}^T \mathbf{y}) : \mathbf{y} \in \mathbb{F}^k \}.$$

Let $\mathcal{I} \subset [n]$ be any set of size $|\mathcal{I}| < \gamma n$.

Then there is an evaluation scheme Φ for $\mathcal{F}$ and $\mathcal{C}$ that tolerates failures in $\mathcal{I}$, and that has bandwidth at most

$$(n - |\mathcal{I}|) \cdot \left(\frac{1}{\varepsilon - \delta} \right) \cdot \lceil \log_2 q \rceil.$$

► **Remark 20.** For constant γ , the requirements on $\varepsilon, \gamma, \delta$ may be satisfied with a choice of $\varepsilon = \gamma + \Theta(1/q)$. Thus, as q grows, ε may approach γ . This means that the trade-off between the rate of the code $(1 - \varepsilon)$ and the fraction of failures tolerated (γ) approaches the Singleton bound, which is optimal (regardless of bandwidth).

6 Conclusion

In this paper we considered low-bandwidth function evaluation on encoded data. Special cases of this problem appear throughout computer science, engineering and cryptography, and we believe that it is valuable to study this problem in generality. We kick off this agenda by studying the problem for *general linear functions* and for *Reed-Solomon codes*, arguably among the most natural classes of functions and codes. However, we hope that this is just the tip of the iceberg. We conclude with several questions left open by this work.

1. Can we develop low-bandwidth evaluation schemes for other classes of functions? (Beyond those mentioned in Remark 6 that are implied by our results?) Low-degree polynomials are perhaps the next most natural class.
2. Can we develop low-bandwidth evaluation schemes for linear functions, for general linear codes? The first part of our framework (in Section 3) applies to general linear codes, but the second part (Section 4) and our main theorem applies only for RS codes.
3. Can we extend our scheme to work in different parameter regimes? In particular, our scheme works with full-length RS codes over extension fields. Work from regenerating codes has shown how to use RS codes as regenerating codes in very different parameter regimes, for example when t is very large [27] or over prime fields [8]. Could these approaches be adapted to low-bandwidth function evaluation?

References

- 1 Ceph authors and contributors. Ceph erasure code documentation. <https://docs.ceph.com/en/latest/rados/operations/erasure-code/>, 2016. Last accessed: 2021.

- 2 Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: optimizations and applications. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2105–2122, 2017.
- 3 Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under dhd. In *Annual International Cryptology Conference*, pages 509–539. Springer, 2016.
- 4 Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 5 Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2019.
- 6 Zachary Charles, Dimitris Papailiopoulos, and Jordan Ellenberg. Approximate gradient coding via sparse random graphs. *arXiv preprint arXiv:1711.06771*, 2017.
- 7 Victor Chen, Elena Grigorescu, and Ronald de Wolf. Error-correcting data structures. *SIAM Journal on Computing*, 42(1):84–111, 2013.
- 8 Roni Con and Itzhak Tamo. Nonlinear repair schemes of reed-solomon codes. *arXiv preprint arXiv:2104.01652*, 2021.
- 9 Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE transactions on information theory*, 56(9):4539–4551, 2010.
- 10 Alexandros G Dimakis, Kannan Ramchandran, Yunnan Wu, and Changho Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, 2011.
- 11 Sanghamitra Dutta, Vivek Cadambe, and Pulkrit Grover. “short-dot”: Computing large linear transforms distributedly using coded short dot products. *IEEE Transactions on Information Theory*, 65(10):6171–6193, 2019.
- 12 Ingerid Fosli, Yuval Ishai, Victor Kolobov, and Mary Wootters. On the download rate of homomorphic secret sharing, 2021. Manuscript.
- 13 Matthew Franklin and Moti Yung. Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710, 1992.
- 14 Venkatesan Guruswami and Mary Wootters. Repairing Reed-Solomon codes. *IEEE transactions on Information Theory*, 63(9):5684–5698, 2017.
- 15 Apache Hadoop. HDFS erasure coding documentation. <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html>, 2015. Last accessed: 2021.
- 16 Wael Halbawi, Navid Azizan, Fariborz Salehi, and Babak Hassibi. Improving distributed gradient descent using reed-solomon codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2027–2031. IEEE, 2018.
- 17 Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.
- 18 Andreas Lenz, Rawad Bitar, Antonia Wachter-Zeh, and Eitan Yaakobi. Function-correcting codes. *arXiv preprint arXiv:2102.03094*, 2021.
- 19 Songze Li and Salman Avestimehr. *Coded Computing: Mitigating Fundamental Bottlenecks in Large-scale Distributed Computing and Machine Learning*. Now Foundations and Trends, 2020.
- 20 Songze Li, Seyed Mohammadreza Mousavi Kalan, A Salman Avestimehr, and Mahdi Soltanolkotabi. Near-optimal straggler mitigation for distributed gradient methods. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 857–866. IEEE, 2018.

- 21 Songze Li, Mohammad Ali Maddah-Ali, Qian Yu, and A Salman Avestimehr. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, 2017.
- 22 Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent ot. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 678–708. Springer, 2021.
- 23 Netanel Raviv, Itzhak Tamo, Rashish Tandon, and Alexandros G Dimakis. Gradient coding from cyclic mds codes and expander graphs. *IEEE Transactions on Information Theory*, 66(12):7475–7489, 2020.
- 24 Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from dcr and applications. *IACR Cryptol. ePrint Arch.*, 2021:274, 2021.
- 25 Karthikeyan Shanmugam, Dimitris S Papailiopoulos, Alexandros G Dimakis, and Giuseppe Caire. A repair framework for scalar mds codes. *IEEE Journal on Selected Areas in Communications*, 32(5):998–1007, 2014.
- 26 Noah Shetty and Mary Wootters. Low-bandwidth recovery of linear functions of reed-solomon-encoded data. *arXiv preprint arXiv:2107.11847*, 2021.
- 27 Itzhak Tamo, Min Ye, and Alexander Barg. The repair problem for reed-solomon codes: Optimal repair of single and multiple erasures with almost optimal node size. *IEEE Transactions on Information Theory*, 65(5):2673–2695, 2018.
- 28 Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376. PMLR, 2017.
- 29 Min Ye and Emmanuel Abbe. Communication-computation efficient gradient coding. In *International Conference on Machine Learning*, pages 5610–5619. PMLR, 2018.
- 30 Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A Avestimehr. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1215–1225. PMLR, 2019.
- 31 Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Coded fourier transform. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 494–501. IEEE, 2017.
- 32 Qian Yu, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4406–4416, 2017.