Coded Computing for Secure Boolean Computations

Chien-Sheng Yang, *Graduate Student Member, IEEE*, and A. Salman Avestimehr, *Fellow, IEEE*

***Abstract*—The growing size of modern datasets necessitates splitting a large scale computation into smaller computations and operate in a distributed manner. Adversaries in a distributed system deliberately send erroneous data in order to affect the computation for their benefit. Boolean functions are the key components of many applications, e.g., verification functions in blockchain systems and design of cryptographic algorithms. We consider the problem of computing a Boolean function in a distributed computing system with particular focus on *security against Byzantine workers*. Any Boolean function can be modeled as a multivariate polynomial with high degree in general. However, the security threshold (i.e., the maximum number of adversarial workers can be tolerated such that the correct results can be obtained) provided by the recent proposed Lagrange Coded Computing (LCC) can be extremely low if the degree of the polynomial is high. We propose three different schemes called *coded Algebraic normal form* (*ANF*), *coded Disjunctive normal form* (*DNF*) and *coded polynomial threshold function* (*PTF*). The key idea of the proposed schemes is to model it as the concatenation of some low-degree polynomials and threshold functions. In terms of the security threshold, we show that the proposed coded ANF and coded DNF are optimal by providing a matching outer bound.**

***Index Terms*—Boolean function, coded computing, distributed computing.**

I. Introduction

With the growing size of modern datasets for applications such as machine learning and data science, it is necessary to partition a massive computation into smaller computations and perform these smaller computations in a distributed manner for improving overall performance [2]. However, distributing the computations to some external entities, which are not necessarily trusted, i.e., adversarial servers make security a major concern [3]–[5]. Thus, it is important to provide security against adversarial workers that deliberately send erroneous data in order to affect the computation for their benefit.

Boolean functions are primarily used in the design of cryptographic algorithms [6]. In particular, computing Boolean functions is one of the key components of blockchains. In the blockchain systems, Boolean functions can be used to represent the verification functions which validate the transactions in the new proposed blocks [7]. Specifically, each node computes function `is_valid_txn` $\in$ {`True`, `False`} to determine whether a transaction is valid or not [8]. Due to the heavy computation cost incurred by validating all the blocks, the nodes with limited resources cannot verify all the blocks independently. To improve the efficiency (e.g., number of transactions verified by the system), the leading solution is via sharding [9] whose idea is to partition the blockchain into sub-chains and the block validations are executed distributively in each node.

In this article, we consider the problem of computing a Boolean function (e.g., block validation) in which the computation is carried out distributively across several workers with particular focus on *security against Byzantine workers*. Specifically, using a master-worker distributed computing system with N workers, the goal is to compute the Boolean function $f : \{0, 1\}^m \to \{0, 1\}$ over a dataset of K samples $X_1, \ldots, X_K$, i.e., $f(X_1), \ldots, f(X_K)$, in which the (encoded) datasets are prestored in the workers such that the computations can be secure against adversarial workers in the system. Especially, we consider the adversarial model in which the malicious workers do not have any computational restriction and are capable of sending erroneous data. To measure the robustness against adversaries of a given scheme S, we use the metric *security threshold* β_S which is defined as the maximum number of adversarial workers that can be tolerated by the master, i.e., the correct results can be recovered even if there are up to β_S adversarial workers.

Any Boolean function can be modeled as an Algebraic normal form (i.e., multivariate polynomial) [6]. Thus, the recently proposed Lagrange Coded Computing (LCC) [10], a universal encoding technique for arbitrary multivariate polynomial computations, can be used to simultaneously alleviate the issues of resiliency, security, and privacy. In overview, for the problem of computing an arbitrary multivariate polynomial $f : \mathbb{V} \to \mathbb{U}$ over a field $\mathbb{F}$, LCC encodes $X_1, \ldots, X_K \in \mathbb{V}$ by evaluating the well-known Lagrange polynomial, and each encoded data is stored in a different worker. The workers then apply the multivariate polynomial of interest f (e.g., Boolean function) on their encoded data and return the computation results back to the master. Since the computation executed in each worker can be viewed as a composition of a multivariate polynomial and a univariate polynomial, the problem becomes a polynomial interpolation with errors and erasures. The master recovers the computation by evaluating the interpolated polynomial at the appropriately chosen points.

Manuscript received August 16, 2020; revised December 4, 2020 and January 21, 2021; accepted January 21, 2021. Date of publication January 29, 2021; date of current version March 16, 2021. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract HR001117C0053; in part by the Army Research Office (ARO) under Award W911NF1810400; in part by NSF under Grant CCF-1703575 and Grant CCF-1763673; and in part by the Office of Naval Research (ONR) under Award N00014-16-1-2189. A part of this paper was presented in ISITA2020 [1]. *(Corresponding author: Chien-Sheng Yang.)*

The authors are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: chienshy@usc.edu; avestimehr@ee.usc.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/JSAIT.2021.3055341, provided by the authors.

Digital Object Identifier 10.1109/JSAIT.2021.3055341

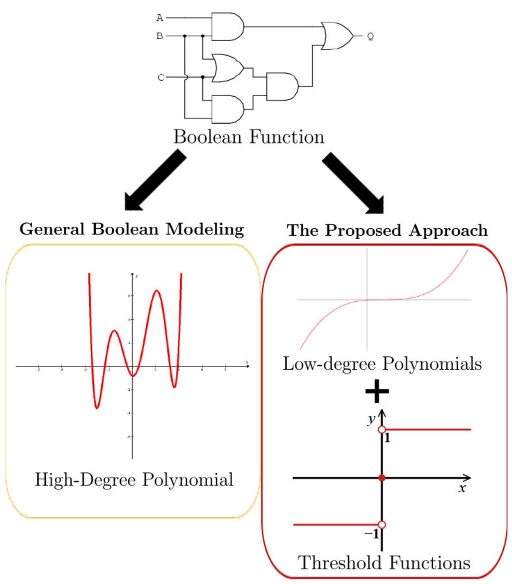

Fig. 1. Modeling the Boolean function as a general polynomial can result in the high-degree difficulty which makes the security threshold low by using LCC encoding. The main idea of our proposed approach is to model it as the concatenation of some low-degree polynomials and the threshold functions.

The security threshold provided by LCC is $\lfloor\frac{N-(K-1)\deg f-1}{2}\rfloor$ (given N and K) which can be extremely low if the degree of corresponding multivariate polynomial $\deg f$ is high (see more details in Section III). Such degree problem can be further amplified in complex Boolean functions whose degree can be high in general. Thus, our main problem is as follows: what is the maximum possible security threshold and the corresponding scheme, given f, N and K?

A. Main Contributions

As main contributions of this article, instead of modeling the Boolean function as a general polynomial, we propose the three schemes modeling it as the concatenation of some low-degree polynomials and the threshold functions (see Figure 1). To illustrate the main idea of the proposed schemes, consider an AND function of three input bits $X[1], X[2], X[3]$ which is formally defined by $f(X) = X[1] \wedge X[2] \wedge X[3]$. The function f can be modeled as a polynomial function (Algebraic normal form) $X[1]X[2]X[3]$ which has a degree of 3. For this polynomial, LCC achieves the security threshold $\lfloor\frac{N-3(K-1)-1}{2}\rfloor$. Instead of directly computing the degree-3 polynomial, our proposed approach is to model it as a linear threshold function $\text{sgn}(X[1]+X[2]+X[3]-\frac{5}{2})$ in which $f(X) = 1$ if and only if $\text{sgn}(X[1]+X[2]+X[3]-\frac{5}{2}) > 0$. Then, a simple linear code (e.g., (N, K) MDS code) can be used for computing the linear function $X[1]+X[2]+X[3]$, which provides the optimal security threshold $\lfloor\frac{N-K}{2}\rfloor$.

We propose three different schemes called *coded Algebraic normal form* (*ANF*), *coded Disjunctive normal form* (*DNF*) and *coded polynomial threshold function* (*PTF*). The idea behind coded ANF (DNF) is to first decompose the Boolean function into some monomials (clauses) and then construct a linear threshold function for each monomial (clause). For both of coded ANF and coded DNF, an (N, K) MDS code is used to encode the datasets. On the other hand, the proposed coded PTF models the Boolean function as a low-degree polynomial threshold function, and LCC is used for the data encoding.

TABLE I
PERFORMANCE COMPARISON OF LCC AND THE PROPOSED THREE SCHEMES FOR THE BOOLEAN FUNCTION $f : \{0, 1\}^m \rightarrow \{0, 1\}$ WHICH HAS THE SPARSITY $r(f)$ AND WEIGHT $w(f)$

	Security Threshold	**Decoding Complexity**
LCC	$\lfloor\frac{N-(K-1)\mathrm{def} f-1}{2}\rfloor$	$O(mN\log^3 N\log\log N)$
Coded ANF	$\lfloor\frac{N-K}{2}\rfloor$	$O(r(f)N\log^2 N\log\log N)$
Coded DNF	$\lfloor\frac{N-K}{2}\rfloor$	$O(w(f)N\log^2 N\log\log N)$
Coded PTF	$\lfloor\frac{N-(K-1)(\lfloor\log_2 w(f)\rfloor+1)-1}{2}\rfloor$	$O(N\log^2 N\log\log N)$
Outer Bound	$\lfloor\frac{N-K}{2}\rfloor$	-

For any general Boolean function f, the proposed coded ANF and coded DNF achieve the security threshold $\lfloor\frac{N-K}{2}\rfloor$, which is independent of $\deg f$. In terms of security threshold, we prove that coded ANF and coded DNF are optimal by deriving a matching theoretical outer bound. To demonstrate the impact of coded ANF and coded DNF, we consider the problem of computing 8-bit S-box in the application of block cyphers using a distributed computing system with 100 workers. We show that coded ANF and coded DNF can significantly improve the security threshold by 150% as compared to LCC.

In Table I, we summarize the performance comparison of LCC and the proposed three schemes in terms of the security threshold and the decoding complexity. As compared to LCC, coded ANF and coded DNF provide the substantial improvement on the security threshold. In particular, coded ANF has the decoding complexity $O(r(f)N\log^2 N\log\log N)$ which works well for the Boolean functions with low sparsity $r(f)$; coded DNF has the decoding complexity $O(w(f)N\log^2 N\log\log N)$ which works well for the Boolean functions with small weight $w(f)$ (see the definitions of $r(f)$ and $w(f)$ in Section II). For the Boolean functions with the polynomial size of $r(f)$ and $w(f)$, coded PTF outperforms LCC by achieving the better security threshold and the almost linear decoding complexity which is independent of m (see more details in Section VI).

Finally, we extend the problem to a more general computation model, i.e., f is a multivariate polynomial function. To resolve the high-degree difficulty arising in computing general polynomials, we propose two schemes: *coded data logarithm* and *coded data augmentation*. By taking the logarithm of original data, the proposed coded data logarithm scheme reduces the degree of polynomial computations, and improves the security threshold as compared to LCC. On the other hand, the proposed coded data augmentation scheme prestores some low-degree monomials in advance to make the polynomial computation's degree reduced.

B. Related Prior Work

Next, we provide a brief literature review that covers two main lines of work: polynomial threshold functions representing Boolean functions, and coded computing.

The expressive power of real polynomial threshold functions for representing Boolean functions has been extensively studied over the decades. The study of representing Boolean functions by polynomial threshold functions was initiated in [11]–[13]. The following works focused largely on the degree of PTF needed to represent a Boolean function (e.g., [14]–[18]), and the density of PTF needed to represent a Boolean function (e.g., [17], [19]–[21]). Polynomials threshold functions also play a vital role in complexity theory and learning theory (e.g., [22], [23]).

Coded computing broadly refers to a family of techniques that utilize coding to inject computation redundancy in order to alleviate the various issues that arise in large-scale distributed computing. In the past few years, coded computing has had a tremendous success in various problems, such as straggler mitigation and bandwidth reduction (e.g., [24]–[35]). Coded computing has also been expanded in various directions, such as heterogeneous networks (e.g., [36]), partial stragglers (e.g., [37]), secure and private computing (e.g., [10], [38]–[44]), distributed optimization (e.g., [45]), federated learning (e.g., [46]–[48]), blockchains (e.g., [7], [49]) and dynamic networks (e.g., [50]–[52]).

So far, research in coded computing has focused on developing frameworks for some linear functions (e.g., matrix multiplications). However, there has been no works prior to our work that consider coded computing for Boolean functions. In this article, we make the substantial progress of improving the security threshold by proposing coded ANF, coded DNF and coded PTF which leverage the idea of the threshold function representation.

Notation: For the Boolean logical operations, we denote the logical operators of AND, OR, XOR and NOT by $\wedge$, $\vee$, $\oplus$ and $\sim$ respectively.

II. System Model

We consider the problem of evaluating a Boolean function $f : \{0,1\}^m \to \{0,1\}$ over a dataset $\vec{X} = (X_1, \ldots, X_K)$, where $X_1, \ldots, X_K$ are m-dimensional vectors over the field $\{0,1\}$. Given a distributed computing environment with a master and N workers, our goal is to compute $f(X_1), \ldots, f(X_K)$.

Each Boolean function $f : \{0,1\}^m \to \{0,1\}$ can be represented by an Algebraic normal form (ANF) [6], [53] as follows:

$$f(X) = \bigoplus_{\mathcal{S}\subseteq[m]} \mu_f(\mathcal{S}) \prod_{j\in\mathcal{S}} X[j] \tag{1}$$

where $X[j]$ is the j-bit of data X and $\mu_f(\mathcal{S}) \in \{0,1\}$ is the ANF coefficient of the corresponding monomial $\prod_{j\in\mathcal{S}} X[j]$. The total degree[1] of the ANF representation of Boolean function f is denoted by deg$f$. We denote the sparsity (number of monomials) of f by $r(f)$, i.e., $r(f) = \sum_{\mathcal{S}\subseteq[m]} \mu_f(\mathcal{S})$. Since each monomial in ANF has the degree up to deg$f$, the total complexity of computing $f(X_1), \ldots, f(X_K)$ via ANF of f is $O(Kr(f)\deg f)$.

[1]The total degree of a multivariate polynomial is the maximum among all the total degrees of its monomials.

Furthermore, we denote the support of f by Supp(f) which is the set of vectors in $\{0,1\}^m$ such that $f = 1$, i.e., Supp(f) $= \{X \in \{0,1\}^m : f(X) = 1\}$. Let $w(f)$ be the weight of Boolean function f, defined by $w(f) = |\text{Supp}(f)|$. Alternatively, each Boolean function f can be represented by a Disjunctive normal form (DNF) as follows:

$$f = T_1 \vee T_2 \vee \cdots \vee T_{w(f)} \tag{2}$$

where each clause T_i has m literals[2] in which each literal corresponds to an input Y_i such that $f(Y_i) = 1$. For example, if $Y_i = 001$, then the corresponding clause is $\sim Y_i[1] \wedge \sim Y_i[2] \wedge Y_i[3]$. Since each clause of DNF has m literals, the total complexity of computing $f(X_1), \ldots, f(X_K)$ via DNF of f is $O(Kmw(f))$.

Prior to computation, each worker has already stored a fraction of the dataset in a possibly coded manner. Specifically, each worker n stores $\tilde{X}_n = g_n(X_1, \ldots, X_K)$, where $g_n : \underbrace{\{0,1\}^m \times \cdots \times \{0,1\}^m}_{K} \to \mathbb{U}$ is the encoding function of worker n and $\mathbb{U}$ is an arbitrary vector space. We restrict our attention to linear encoding schemes, which guarantee low encoding complexity. Each worker n computes $h(\tilde{X}_n)$ and returns the result back to the master, in which h is the multivariate polynomial function decided by the master and $f(X)$ is function of $h(X)$. Then, the master aggregates the results from the workers until it receives a *decodable* set of local computations. We say a set of computations is decodable if $h(X_1), \ldots, h(X_K)$ can be obtained by computing decoding functions over the received results.

More concretely, given any subset of workers that return the computing results (denoted by $\mathcal{K}$), the master computes $v_{\mathcal{K}}(\{h(\tilde{X}_n)\}_{n\in\mathcal{K}})$, where each $v_{\mathcal{K}}$ is a deterministic function. We refer to the $v_{\mathcal{K}}$'s as decoding functions. Finally, the master computes $f(X_1), \ldots, f(X_K)$ based on $h(X_1), \ldots, h(X_K)$.

In particular, we focus on finding the scheme $(\vec{g}, h)$ to be robust to as many adversarial workers as possible in the system where $\vec{g} = (g_1, \ldots, g_N)$ is the collection of encoding functions. To measure the robustness against adversaries of a given scheme, we use the metric *security threshold* defined as follows:

Definition 1 (Security Threshold): For an integer b, we say a scheme S is b-secure if the master can be robust against b adversaries, i.e., the master can recover all the correct results even if up to b workers return arbitrarily erroneous results. The security threshold, denoted by β_S, is the maximum value of b such that a scheme S is b-secure, i.e.,

$$\beta_S \triangleq \sup\{b : S \text{ is } b\text{-secure}\}. \tag{3}$$

Based on the above system model, the problem is now formulated as: *What is the scheme which achieves the optimal security threshold with low decoding complexity?*

Remark 1: To see how much computation cost that the master can save using a given scheme, it is important to compare the total complexity of computing K evaluations $f(X_1), \ldots, f(X_K)$ (by the master itself) with the complexity incurred by the scheme. Since the encoding process of a

[2]A literal is a Boolean variable or the complement of a Boolean variable.

scheme is only executed once before starting any computations, we focus on the decoding complexity which is the main cost incurred by a scheme throughout this article.

Remark 2: To see how the distributed Boolean computation is applicable to a sharded blockchain system, we can consider a blockchain system `PolyShard` [7] which is implemented distributedly over some untrusted nodes. At each time epoch, each node stores a coded version of sub-chain and computes a validation function directly on the coded sub-chain and a coded block (generated by computing an encoding function on the incoming blocks). After the computations, each node broadcasts the computed result to all other nodes. Then, each node computes the decoding function on the received computation results to reduce the desired validation result and determines the validity of block. That is, each node plays the role of a master node after the procedure of broadcasting. When there is a new participant joining the network, a new coded sub-chain can be generated and stored in this new node. When there is a participant leaving the network, the blockchain with remaining nodes can still work since each node stores a coded sub-chain and the system can follow the same procedure for the block validations.

III. Overview of Lagrange Coded Computing

In this section, we consider the recently proposed Lagrange Coded Computing (LCC) [10], which is a universal encoding technique for the class of multivariate polynomial functions. Then, we show how it works for our problem.

Since Lagrange coded computing requires the underlying field size to be at least the number of workers N, we first extend the field size of $\{0, 1\}$ such that the size of extension field is at least the number of workers N. More specifically, we embed each bit $X_k[j] \in \{0, 1\}$ of data X_k into a binary extension field $\mathbb{F}_{2^t}$ such that with $2^t \geq N$. The embedding $\bar{X}_k[j] \in \mathbb{F}_{2^t}$ of the bit $X_k[j]$ is generated such that

$$\bar{X}_k[j] = \begin{cases} \underbrace{00\cdots0}_{t}, & X_k[j] = 0, \\ \underbrace{00\cdots0}_{t-1}1, & X_k[j] = 1. \end{cases} \tag{4}$$

Note that over extension field the output of Boolean function f is $\underbrace{00\cdots0}_{t}$ if the original result is 0; $\underbrace{00\cdots0}_{t-1}1$ if the original result is 1.

For the data encoding by using LCC, we first select K distinct elements $\beta_1, \beta_2, \ldots, \beta_K$ from the binary extension field $\mathbb{F}_{2^t}$, and let u be the respective *Lagrange interpolation polynomial*:

$$u(z) \triangleq \sum_{k=1}^{K} \bar{X}_k \prod_{l\in[K]\setminus\{k\}} \frac{z - \beta_l}{\beta_k - \beta_l}, \tag{5}$$

where $u : \mathbb{F}_{2^t} \to \mathbb{F}_{2^t}^m$ is a polynomial of degree $K - 1$ such that $u(\beta_k) = \bar{X}_k$. Then we can select distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbb{F}_{2^t}$, and encode $\bar{X}_1, \ldots, \bar{X}_K$ to $\tilde{X}_n = u(\alpha_n)$ for all $n \in [N]$, i.e.,

$$\tilde{X}_n = u(\alpha_n) \triangleq \sum_{k=1}^{K} \bar{X}_k \prod_{l\in[K]\setminus\{k\}} \frac{\alpha_n - \beta_l}{\beta_k - \beta_l}. \tag{6}$$

Each worker $n \in [N]$ stores $\tilde{X}_n$ locally. Following the above data encoding, each worker n computes function f on $\tilde{X}_n$ and sends the result back to the master upon its completion. Since the computation is over the extension field, the complexity at each worker is $O(tr(f)\mathrm{deg}f)$.

After receiving results from all the workers, the master can obtain all coefficients of $f(u(z))$ by applying Reed-Solomon decoding [54], [55]. Having this polynomial, the master evaluates it at β_k for every $k \in [K]$ to obtain $f(u(\beta_k)) = f(\bar{X}_k)$. The complexity of decoding a length-N Reed-Solomon code with dimension $(K-1)\mathrm{deg}f+1$ for one symbol over the extension field is $O(tN\log^2 N \log\log N)$. To have a sufficiently large field for LCC, we pick $t = \lceil \log N \rceil$. Since there are m symbols in each $\tilde{X}_n$, the decoding process by the master requires complexity $O(mN\log^3 N \log\log N)$.

In the following, we present the security threshold provided by LCC. By [10], to be robust to b adversarial workers (given N and K), LCC requires $N \geq (K-1)\mathrm{deg}f + 2b + 1$; i.e., LCC achieves the security threshold

$$\beta_{\text{LCC}} = \left\lfloor \frac{N - (K-1)\mathrm{deg}f - 1}{2} \right\rfloor. \tag{7}$$

The security threshold achieved by LCC depends on the degree of function f, i.e., the security guarantee is highly degraded if f has high degree. To mitigate such degree effect, we model the Boolean function as the concatenation of some low-degree polynomials and the threshold functions by proposing three schemes in the following sections.

IV. Scheme 1: Coded Algebraic Normal Form

In this section, we propose a coding scheme called *coded Algebraic normal form* (*ANF*) which computes the ANF representations of Boolean function by the linear threshold functions (LTF) and a simple linear code is used for the data encoding. We start with an example to illustrate the idea of coded ANF.

Example 1: We consider a function which has an ANF representation defined as follows:

$$f(X) = X[1]X[2] \cdot X\left[\frac{m}{2}\right]. \tag{8}$$

Then, we define a linear function over real field as follows:

$$L(X) = \sum_{j=1}^{\frac{m}{2}} X[j] \tag{9}$$

with a bias term $B = -\frac{m}{2} + \frac{1}{2}$, where $L(X) + B = \frac{1}{2}$ if and only if $f(X) = 1$. Otherwise, $L(X) + B \leq -\frac{1}{2}$. Thus, we can compute $f(X)$ by computing its corresponding linear threshold function $\mathrm{sgn}(L(X) + B)$, i.e., $f(X) = 1$ if $\mathrm{sgn}(L(X) + B) = 1$; otherwise, $f(X) = 0$ if $\mathrm{sgn}(L(X) + B) = -1$. Unlike computing the function $f(X)$ with the degree $\frac{m}{2}$ which results in low security threshold, computing the linear function $L(X)$ allows us to apply a linear code on the computations which can lead to a much higher security threshold.

A. Formal Description of Coded ANF

Given the ANF representation defined in (1), we now present the proposed coded ANF scheme in the following. For each monomial $\prod_{j\in\mathcal{S}} X[j]$ such that $\mu_f(\mathcal{S}) = 1$, we define a linear function $L_{\mathcal{S}} : \mathbb{R}^m \to \mathbb{R}$ and a bias term $B_{\mathcal{S}} \in \mathbb{R}$ as follows[3]:

$$L_{\mathcal{S}}(X) = \sum_{j\in\mathcal{S}} X[j], \quad B_{\mathcal{S}} = -|\mathcal{S}| + \frac{1}{2}. \tag{10}$$

It is clear that $L_{\mathcal{S}}(X) + B_{\mathcal{S}} = \frac{1}{2}$ if and only if $\prod_{j\in\mathcal{S}} X[j] = 1$. Otherwise, $L_{\mathcal{S}}(X)+B_{\mathcal{S}} \le -\frac{1}{2}$. Thus, there are $r(f)$ constructed linear threshold functions, and each monomial $\prod_{j\in\mathcal{S}} X[j]$ can be computed by its corresponding linear threshold function $\mathrm{sgn}(L_{\mathcal{S}}(X) + B_{\mathcal{S}})$.

By considering each bit in real field, the master encodes $X_1, X_2, \ldots, X_K$ to $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_N$ using an (N, K) MDS code. Each worker $n \in [N]$ stores $\tilde{X}_n$ locally. Each worker $n\in[N]$ computes the functions $\{L_{\mathcal{S}}(\tilde{X}_n)\}_{\{\mathcal{S}\subseteq[m],\mu_f(\mathcal{S})=1\}}$ and then sends the results back to the master. After receiving the results from the workers, the master first recovers $L_{\mathcal{S}}(X_k)$ for each $k \in [K]$ and each $\mathcal{S} \in \{\mathcal{G} : \mathcal{G} \subseteq [m], \mu_f(\mathcal{G}) = 1\}$. Then, the master has $\prod_{j\in\mathcal{S}} X_k[j] = 1$ if $\mathrm{sgn}(L_{\mathcal{S}}(X_k) + B_{\mathcal{S}}) = 1$; $\prod_{j\in\mathcal{S}} X_k[j]=0$ if $\mathrm{sgn}(L_{\mathcal{S}}(X_k) + B_{\mathcal{S}}) = -1$. Lastly, the master recovers $f(X_1), \ldots, f(X_K)$ by summing the monomials. Since each of $r(f)$ linear functions has up to m variables, the complexity at each worker is $O(mr(f))$.

Remark 3: We can demonstrate the decodability of $\{L_{\mathcal{S}}(\tilde{X}_n)\}_{n\in[N]}$'s by converting our problem to the distributed matrix-matrix multiplications as follows. Computing $\{L_{\mathcal{S}}(X_k)\}_{k\in[K]}$ for each $\mathcal{S}$ is equivalent to computing K matrix-matrix multiplications $X_1A, X_2A, \ldots, X_KA$ ($X_1, \ldots, X_K$ are considered as row vectors) where A is an m by $|\mathcal{S}|$ matrix and each column of matrix A is the coefficients of $X[j]$'s in the corresponding $L_{\mathcal{S}}(X)$. Similarly, computing $\{L_{\mathcal{S}}(\tilde{X}_n)\}_{n\in[N]}$ for the corresponding $\mathcal{S}$ is equivalent to computing N matrix-matrix multiplications $\tilde{X}_1A, \tilde{X}_2A, \ldots, \tilde{X}_NA$. Therefore, our problem can be converted to the coded distributed matrix-matrix multiplication in which an (N, K) MDS code is used to each element of the matrices $X_1, \ldots, X_K$ and the encoded matrices $\tilde{X}_1, \ldots, \tilde{X}_N$ are obtained. In [24], it is shown that matrix multiplications $X_1A, X_2A, \ldots, X_KA$ can be recovered from any K out of N coded results $\tilde{X}_1A, \ldots \tilde{X}_NA$ by the MDS property and the linear property of matrix-matrix multiplications. In our problem, we deal with adversarial workers which are treated as errors. Since the system can be robust to $N - K$ erasures, one can show that the system can be robust to $\lfloor\frac{N-K}{2}\rfloor$ errors (adversaries) by Lemma 3 proved in [35].

B. Security Threshold of Coded ANF

To decode the (N, K) MDS code, coded ANF applies Reed-Solomon decoding. Successful decoding requires the number of errors of computation results such that $N \ge K + 2b$. The following theorem shows that the security threshold provided by coded ANF is $\lfloor\frac{N-K}{2}\rfloor$ which is independent of $\deg f$.

Theorem 1: Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, the proposed coded ANF can be robust to b adversaries for computing $\{f(X_k)\}_{k=1}^K$ for any Boolean function f, as long as

$$N \ge K + 2b; \tag{11}$$

i.e., coded ANF achieves the security threshold

$$\beta_{\mathrm{ANF}} = \left\lfloor \frac{N-K}{2} \right\rfloor. \tag{12}$$

Whenever the master receives N results from the workers, the master decodes the computation results using a length-N Reed-Solomon code for each of $r(f)$ linear functions which incurs the total complexity $O(r(f)N\log^2 N \log\log N)$. Computing all the monomials via the signs of corresponding linear threshold functions incurs the complexity $O(Nr(f))$. Lastly, computing $f(X_1), \ldots, f(X_K)$ by summing the monomials incurs the complexity $O(Nr(f))$ since there are $r(f) - 1$ additions in function f. Thus, the total complexity of decoding step is $O(r(f)N\log^2 N \log\log N)$ which works well for small $r(f)$. Note that the operation of this scheme is over real field whose size does not scale with size of m.

V. Scheme 2: Coded Disjunctive Normal Form

In this section, we propose a coding scheme called *coded Disjunctive normal form* (*DNF*) which computes the DNF representations of Boolean function by LTFs and a simple linear code is used for the data encoding. We start with an example to illustrate the idea behind coded DNF.

Example 2: Consider a function which has an ANF representation defined as follows:

$$f(X) = (X[1]\cdots X[m]) \oplus (X[1] \oplus 1)\cdots(X[m] \oplus 1)$$

which has the degree $\deg f = m - 1$ and the number of monomials $r(f) = 2^m - 1$. Alternatively, this function has a DNF representation as follows:

$$f(X) = (X[1] \wedge \cdots \wedge X[m]) \vee (\sim X[1] \wedge \cdots \wedge \sim X[m])$$

which has the weight $w(f) = 2$.

For the clause $X[1]\wedge\cdots\wedge X[m]$, we define a linear function over real field as follow:

$$L_1(X) = X[1] + \cdots + X[m] \tag{13}$$

with a bias term $B_1 = -m + \frac{1}{2}$, where $X[1] \wedge \cdots \wedge X[m] = 1$ if and only if $L_1(X) + B_1 = \frac{1}{2}$. Otherwise, $L_1(X) + B_1 \le -\frac{1}{2}$. Similarly, for the clause $\sim X[1] \wedge \cdots \wedge \sim X[m]$, we define a linear function over real field as follows:

$$L_2(X) = -X[1] - \cdots - X[m] \tag{14}$$

with a bias $B_2 = \frac{1}{2}$, where $\sim X[1] \wedge \cdots \wedge \sim X[m] = 1$ if and only if $L_2(X) + B_2 = \frac{1}{2}$. Otherwise, $L_2(X) + B_2 \le -\frac{1}{2}$. Therefore, we can compute $f(X)$ by computing $\mathrm{sgn}(L_1(X) + B_1)$ and $\mathrm{sgn}(L_2(X) + B_2)$, i.e., $f(X) = 1$ if at least one of $\mathrm{sgn}(L_1(X)+B_1)$ and $\mathrm{sgn}(L_2(X)+B_2)$ is equal to 1. Otherwise,

[3]The linear threshold function defined in (10) is adapted from the degree-1 polynomial threshold function $p(X) = \sum_{j=1}^m Z[j]X[j] - m + \frac{1}{2}$ considered in [17] where $X \in \{-1, 1\}^m$ and $p(X) > 0$ iff $X = Z$. Since the Boolean domain considered in [17] is $\{-1, 1\}$ instead of $\{0, 1\}$ and all the bits are taken into account in $p(X)$, we define (10) by letting $Z[j] = 0, \forall j \notin \mathcal{S}$ and the bias term to be $-|\mathcal{S}| + \frac{1}{2}$ such that only the bits $X[j], \forall j \in \mathcal{S}$ in the domain $\{0, 1\}$ are taken into account in (10).

$f(X) = 0$. Unlike directly computing the function $f(X)$ with the degree of $m-1$, computing the linear functions $L_1(X)$ and $L_2(X)$ allows us to apply a linear code on the computations.

A. Formal Description of Coded DNF

Given the DNF representation defined in (2), we now present the proposed coded DNF scheme in the following. For each clause T_i with the corresponding input $Y_i \in \text{Supp}(f)$ such that $f(Y_i) = 1$, we define a linear function $L_i : \mathbb{R}^m \to \mathbb{R}$ and a bias term $B_i \in \mathbb{R}$ as follows[4]:

$$L_i(X) = \sum_{j=1}^{m} Z_i[j]X[j], \quad B_i = -\sum_{j=1}^{m} Y_i[j] + \frac{1}{2} \tag{15}$$

where

$$Z_i[j] = \begin{cases} 1, & \text{if } Y_i[j] = 1 \\ -1, & \text{if } Y_i[j] = 0. \end{cases} \tag{16}$$

It is clear that $L_i(Y_i) + B_i = \frac{1}{2}$ and $L_i(X) + B_i \leq -\frac{1}{2}$ for all other inputs $X \neq Y_i$. Thus, there are $w(f)$ constructed linear threshold functions, and each clause T_i can be computed by its corresponding linear threshold function $\text{sgn}(L_i(X) + B_i)$.

By considering each bit over real field, the master encodes $X_1, X_2, \ldots, X_K$ to $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_N$ using an (N, K) MDS code. Each worker $n \in [N]$ stores $\tilde{X}_n$ locally. Each worker n computes the functions $L_1(\tilde{X}_n), \ldots, L_{w(f)}(\tilde{X}_n)$ and then sends the results back to the master. After receiving the results from the workers, the master first recovers $L_i(X_k)$ for each $i \in [w(f)]$ and each $k \in [K]$ via MDS decoding. Then, the master has $T_i(X_k) = 1$ if $\text{sgn}(L_i(X_k) + B_i) = 1$; otherwise $T_i(X_k) = 0$. Lastly, the master has $f(X_k) = 1$ if at least one of $T_1(X_k), \ldots, T_{w(f)}(X_k)$ is equal to 1. Otherwise, $f(X_k) = 0$. Since each of $w(f)$ linear functions has m variables, the complexity at each worker is $O(mw(f))$.

B. Security Threshold of Coded DNF

Similar to coded ANF deploying Reed-Solomon code for the decoding process, we have the following theorem to show that the security threshold provided by coded DNF is $\lfloor \frac{N-K}{2} \rfloor$ which is independent of deg$f$.

Theorem 2: Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, the proposed coded DNF can be robust to b adversaries for computing $\{f(X_k)\}_{k=1}^{K}$ for any Boolean function f, as long as

$$N \geq K + 2b; \tag{17}$$

i.e., coded DNF achieves the security threshold

$$\beta_{\text{DNF}} = \left\lfloor \frac{N-K}{2} \right\rfloor. \tag{18}$$

Upon receiving N results from the workers, the master decodes the computation results using a length-N Reed-Solomon code for each of $w(f)$ linear functions which incurs the total complexity $O(w(f)N\log^2 N \log\log N)$. Computing all the clauses via the signs of corresponding linear threshold functions incurs the complexity $O(Nw(f))$. Lastly, computing $f(X_1), \ldots, f(X_K)$ by checking all the clauses requires the complexity $O(Nw(f))$. Thus, the total complexity of decoding step is $O(w(f)N\log^2 N \log\log N)$ which works well for small $w(f)$.

Remark 4: Learning the DNF representation of a Boolean function is an intensively studied problem in computational learning theory and is hard in general [56]. Thus, people focus on some more tractable classes of functions, e.g., $O(\log n)$-term DNF is considered in PAC learning literature [57], which well motivates our proposed coded DNF.

Remark 5: Although both coded ANF and coded DNF achieve the security threshold $\lfloor \frac{N-K}{2} \rfloor$, coded ANF has the decoding complexity $O(r(f)N\log^2 N \log\log N)$ and coded DNF has the decoding complexity $O(w(f)N\log^2 N \log\log N)$. Based on the sparsity $r(f)$ and the weight $w(f)$, one can choose either one of two schemes that has a smaller decoding complexity. When $r(f)$ is smaller than $w(f)$, coded ANF should be chosen. One the contrary, we can choose coded DNF.

VI. Scheme 3: Coded Polynomial Threshold Function

In this section, we propose a coding scheme called *coded polynomial threshold function* (*PTF*) which computes the DNF representations of Boolean function by PTFs and LCC is used for the data encoding.

A. Formal Description of Coded PTF

Given the DNF representation defined in (2), we now present coded PTF. Following the construction proposed in [17], [56], we now construct a polynomial threshold function $\text{sgn}(P(X))$ for computing $f(X)$ where $P : \mathbb{R}^m \to \mathbb{R}$ is a polynomial function with the degree at most $\lfloor \log_2 w(f) \rfloor + 1$. The construction of such PTF has the following steps.

1) *Decision Tree Construction:* We construct an $w(f)$-leaf decision tree over variables $X[1], \ldots, X[m]$ such that each input in Supp(f) arrives at a different leaf. Such a tree can be always constructed by a greedy algorithm. Let ℓ_i be a leaf of this tree in which Y_i reaches leaf ℓ_i. We label ℓ_i with the linear threshold function $\text{sgn}(L_i(X) + B_i)$ where $L_i(X)$ and B_i are defined in (15). The constructed decision tree, in which internal nodes are labeled with variables and leaves are labeled with linear threshold functions, computes exactly f.
2) *Decision List Construction:* For this $w(f)$-leaf decision tree, we construct an equivalent $\lfloor \log_2 w(f) \rfloor$-decision list. Following from the definition that the rank of an $w(f)$-leaf tree is at most $\lfloor \log_2 w(f) \rfloor$. We find a leaf in the decision tree at distance at most $\lfloor \log_2 w(f) \rfloor$ from the root, and place the literals along the path to the leaf as a monomial at the top of a new decision list. We then remove the leaf from the tree, creating a new decision tree with one fewer leaf, and repeat this process [58]. Without loss of generality, we let ℓ_i be the i-th removed leaf in the process of list construction with the corresponding monomial C_i of at most $\lfloor \log_2 w(f) \rfloor$ variables. The constructed list is defined as if $C_1(X) = 1$ then

[4]Similar to the linear threshold function defined in (10), we define (15) by adjusting the bias term such that the threshold function can work in the domain of {0, 1}.

output $\frac{1+\mathrm{sgn}(L_1(X)+B_1)}{2}$; else if $C_2(X) = 1$ then output $\frac{1+\mathrm{sgn}(L_2(X)+B_2)}{2}$; $\cdots$ else if $C_{w(f)}(X) = 1$ then output $\frac{1+\mathrm{sgn}(L_{w(f)}(X)+B_{w(f)})}{2}$.

3) *Polynomial Threshold Function Construction:* Having the constructed decision list, we now construct the polynomial function $P(X)$ with degree of at most $\lfloor \log w(f) \rfloor + 1$ as follows:

$$P(X) = A_1 C_1(X)(L_1(X) + B_1) + \cdots + A_{w(f)} C_{w(f)}(X)\big(L_{w(f)}(X) + B_{w(f)}\big)$$

where $A_1 \gg A_2 \gg A_3 \cdots \gg A_m > 0$ are appropriately chosen positive values.

After constructing the corresponding PTF $\mathrm{sgn}(P(X))$ for Boolean function $f(X)$, the procedure of computations is as follows. By considering each bit over real field, the master encodes $X_1, X_2, \ldots, X_K$ to $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_N$ using LCC. Each worker $n \in [N]$ stores $\tilde{X}_n$ locally. Each worker n computes the function $P(\tilde{X}_n)$ and then sends the result back to the master. After receiving the results from the workers, the master first recovers $P(X_1), \ldots, P(X_K)$ via LCC decoding. Then, the master has $f(X_k) = 1$ if $\mathrm{sgn}(P(X_k)) = 1$; otherwise $f(X_k) = 0$. Since $C_i(X)$'s are monomials with the degree of at most $\lfloor \log_2 w(f) \rfloor$, computing $A_i C_i(X)$ incurs the complexity $O(\lfloor \log_2 w(f) \rfloor)$. Also, computing $L_i(X) + B_i$ incurs the complexity $O(m)$. Thus, computing function $P(X)$ at each worker incurs the total complexity $O(w(f)(\lfloor \log_2 w(f) \rfloor + m))$.

B. Security Threshold of Coded PTF

Since $P(X)$ has degree of at most $\lfloor \log_2 w(f) \rfloor + 1$, to be robust to b adversaries, LCC requires the number of workers N such that $N \geq (K-1)(\lfloor \log_2 w(f) \rfloor + 1) + 2b + 1$. Then, we present the security threshold provided by coded PTF in the following theorem.

Theorem 3: Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, the proposed coded polynomial threshold function can be robust to b adversaries for computing $\{f(X_k)\}_{k=1}^K$ for any Boolean function f, as long as

$$N \geq (K-1)\big(\lfloor \log_2 w(f) \rfloor + 1\big) + 2b + 1; \tag{19}$$

i.e., coded PTF achieves the security threshold

$$\beta_{\mathrm{PTF}} = \left\lfloor \frac{N - (K-1)\big(\lfloor \log_2 w(f) \rfloor + 1\big) - 1}{2} \right\rfloor. \tag{20}$$

Whenever the master receives N results from the workers, the master decodes the computation results using a length-N Reed-Solomon code for the polynomial function which incurs the total complexity $O(N \log^2 N \log\log N)$. Lastly, computing $f(X_1), f(X_2), \ldots, f(X_K)$ by checking the signs requires the complexity $O(N)$. Thus, the total complexity of decoding step is $O(N \log^2 N \log\log N)$.

In the following example, we show that coded PTF outperforms LCC for the Boolean functions with the polynomial size of $r(f)$ and $w(f)$.

Example 3: Consider a function which has an ANF representation defined as follows:

$$f(X) = (X[1] \oplus X[2]) \cdots \big(X[2m'-1]\big) \oplus X[2m']\big) \times X\big[2m'+1\big] \cdots X[m] \tag{21}$$

where $m' = \lfloor \log_2 m^2 \rfloor$. Note that here we focus on the case that m is large enough such that $m > m' = \lfloor \log_2 m^2 \rfloor$. The function f has the degree of $m - \lfloor \log_2 m^2 \rfloor$, the sparsity of $\approx m^2$ and the weight of $\approx m^2$.

For the Boolean function considered in Example 3, coded PTF achieves the security threshold $\lfloor \frac{N-(K-1)(\lfloor \log_2 m^2 \rfloor + 1) - 1}{2} \rfloor$ which is greater than the security threshold $\lfloor \frac{N-(K-1)(m - \lfloor \log_2 m^2 \rfloor) - 1}{2} \rfloor$ provided by LCC. Although coded ANF and coded DNF achieve security threshold $\lfloor \frac{N-K}{2} \rfloor$ but they require decoding complexity $O(m^2 N \log^2 N \log\log N)$ which has the order of m^2, i.e., they only work for small m. With the security slightly worse than coded ANF and coded DNF, coded PTF achieves the better decoding complexity which is independent of m, i.e., coded PTF can work for large m.

C. Coded D-Partitioned PTF

In this section, we extend coded PTF by proposing *coded D-partitioned polynomial threshold function* whose idea is to partition the Boolean function into some DNFs and construct their corresponding PTFs with low-degree. It allows us to apply LCC on the corresponding low-degree PTFs for improving the security threshold.

Given the DNF representation defined in (2) of Boolean function f and an integer D ($1 \leq D \leq w(f)$), we partition the DNF representation of f to D different DNF representations as follows:

$$f = \mathcal{G}_1 \vee \mathcal{G}_2 \vee \cdots \vee \mathcal{G}_D \tag{22}$$

where each $\mathcal{G}_d$ includes $\frac{w(f)}{D}$ clauses of m literals, e.g.,

$$\mathcal{G}_1 = T_1 \vee \cdots \vee T_{\frac{w(f)}{D}}. \tag{23}$$

Thus, we have that each $\mathcal{G}_d$ is a Boolean function with weight of $\frac{w(f)}{D}$. By the PTF construction described in Section VI-A, each Boolean function $\mathcal{G}_d$ can be computed by a PTF $\mathrm{sgn}(P_d(X))$ where $P_d(X)$ has degree of at most $\lfloor \log_2 \frac{w(f)}{D} \rfloor + 1$.

Similar to coded PTF using LCC for data encoding, each worker $n \in [N]$ stores $\tilde{X}_n$ locally. Each worker n computes the function $P_1(\tilde{X}_n), \ldots, P_D(\tilde{X}_n)$ and then sends the results back to the master. Upon receiving the results from the workers, the master first recovers $P_d(X_1), \ldots, P_d(X_K)$ for each d via LCC decoding. Then, the master has $f(X_k) = 1$ if at least one of $\mathrm{sgn}(P_1(X_k)), \ldots, \mathrm{sgn}(P_D(X_k))$ is equal to 1. Otherwise, $f(X_k) = 0$. Similar to coded PTF, computing D polynomial functions with the degree up to $\lfloor \log_2 \frac{w(f)}{D} \rfloor + 1$ at each worker incurs the complexity $O(w(f)(\lfloor \log_2 \frac{w(f)}{D} \rfloor + m))$.

Since each $P_d(X)$ has degree of at most $\lfloor \log_2 \frac{w(f)}{D} \rfloor + 1$, to be robust to b adversaries, LCC requires the number of workers N such that $N \geq (K-1)(\lfloor \log_2 \frac{w(f)}{D} \rfloor + 1) + 2b + 1$. Formally, we have the following theorem.

Theorem 4: Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, the proposed coded D-partitioned polynomial threshold function can be robust to b adversaries for computing $\{f(X_k)\}_{k=1}^{K}$ for any Boolean function f, as long as

$$N \geq (K-1)\left(\left\lfloor \log_2 \frac{w(f)}{D} \right\rfloor + 1\right) + 2b + 1; \tag{24}$$

i.e., coded D-partitioned PTF achieves the security threshold

$$\beta_{\text{PTF}}(D) = \left\lfloor \frac{N - (K-1)\left(\left\lfloor \log_2 \frac{w(f)}{D} \right\rfloor + 1\right) - 1}{2} \right\rfloor. \tag{25}$$

Whenever the master receives N results from the workers, the master decodes the computation results using a length-N Reed-Solomon code for D constructed polynomial function which incurs the total complexity $O(DN \log^2 N \log\log N)$. Then, computing $f(X_1), f(X_2), \ldots, f(X_K)$ by checking the signs and OR operations requires the complexity $O(DN)$. Thus, the total complexity of decoding step is $O(DN \log^2 N \log\log N)$.

Remark 6: The proposed coded D-partitioned PTF characterize a tradeoff between the security threshold and the decoding complexity. For each chosen $D(1 \leq D \leq w(f))$, the pair of the security threshold and the decoding complexity $(\lfloor \frac{N-(K-1)(\lfloor \log_2 \frac{w(f)}{D} \rfloor+1)-1}{2} \rfloor, DN \log^2 N \log\log N)$ can be achieved by the proposed coded D-partitioned PTF. In particular, the proposed coded DNF and coded PTF schemes correspond to the two extreme points of this tradeoff that minimize the security threshold and the decoding complexity respectively. Coded DNF corresponds to the point $D = 1$, i.e., no partition performed. On the other hand, coded corresponds to the point $D = w(f)$, i.e., each DNF after partition process only contains one vector in $\{0, 1\}^m$. Thus, coded D-partitioned PTF generalizes our previously proposed coded DNF and coded PTF, and allows to systematically operate at any points on this tradeoff.

Remark 7: The total complexity of computing K evaluations $f(X_1), \ldots, f(X_K)$ via ANF is $O(Kr(f)\deg f)$. Thus, it is more efficient to use coded ANF than computing all the evaluations at the master when $\deg f > \frac{N}{K} \log^2 N \log\log N$. On the other hand, since computing $f(X_1), \ldots, f(X_K)$ via DNF incurs the total complexity $O(Kmw(f))$, we can conclude that it is more efficient to use coded DNF when $m > \frac{N}{K} \log^2 N \log\log N$. When $mw(f) > \frac{DN}{K} \log^2 N \log\log N$, coded D-partitioned PTF is more efficient than computing all the evaluations at the master.

VII. Matching Outer Bound for Coded ANF and Coded DNF

In this section, we show that coded ANF and coded DNF are optimal in terms of the security threshold. We start by defining the recovery threshold and the hamming distance of a scheme as follows:

Definition 2: For any integer k, we say a scheme is k-recoverable if the master can recover $h(X_1), \ldots, h(X_K)$ given the computing results from any k workers. We define the recovery threshold of a scheme $(\vec{g}, h)$, denoted by $K(\vec{g}, h)$, as the minimum integer k such that scheme $(\vec{g}, h)$ is k-recoverable.

Definition 3: We define the Hamming distance of any scheme $(\vec{g}, h)$, denoted by $d(\vec{g}, h)$, as the maximum integer d such that for any pair of input dataset whose computation results $h(X_1), \ldots, h(X_K)$ are different, at least d workers compute different values of $h(\tilde{X}_n)$.

We prove the matching outer bound for coded ANF and coded DNF by the following theorem whose proof can be found in Appendix A.

Theorem 5: For a distributed computing problem of computing Boolean function f using N workers over a dataset $X = (X_1, \ldots, X_K)$, any scheme $(\vec{g}, h)$ can achieve the security threshold up to

$$\beta^* = \left\lfloor \frac{N-K}{2} \right\rfloor. \tag{26}$$

By Theorem 5, we have shown that the proposed coded ANF and coded DNF schemes are optimal in terms of the security threshold.

VIII. Application to Cryptography

To demonstrate the impact of the proposed schemes, we consider a cryptosystem which is designed to enable two parties to securely communicate over an insecure channel [59]. In a cryptosystem, the plaintext is encrypted to the cyphertext before the communication from one user to the another user, e.g., one user of a party shares the same secret key with the user of another party to communicate secretly. Since the security of symmetric cryptosystems is strongly influenced by Boolean functions, many properties of Boolean functions must be utilized (e.g., high nonlinearity, high algebraic degree, and etc) in order to resist the known mathematical attacks. More specifically, a cipher must not be well-approximated by linear functions to be secure against linear attacks [60]. High algebraic degree of Boolean function increases the linear complexity in block ciphers and result in more complicated systems of equations describing the cipher which make structural attacks of the cipher more difficult [61].

In particular, we focus on one of subclasses of symmetric key cryptosystem: block cyphers. As the non-linear component in most block ciphers, *S-boxes* are one of the most important building blocks in symmetric cryptography and chosen to be cryptographically strong enough against the attacks. Formally, an S-box $s : \{0, 1\}^m \to \{0, 1\}^m$ is represented by a collection of m Boolean functions of m input bits, and each Boolean function is one of the coordinates of function s. Please see Table II for an example of a 4-bit S-box.

Each coordinate of $s(X)$ presented in the example in Table II can be represented by a degree-3 ANF representation as follows:

$$\begin{aligned} s(X)_1 = 1 &\oplus X[1] \oplus X[3] \oplus X[4] \oplus X[2]X[3] \oplus X[2]X[4] \\ &\oplus X[3]X[4] \oplus X[1]X[3]X[4] \oplus X[2]X[3]X[4] \end{aligned} \tag{27}$$

$$\begin{aligned} s(X)_2 = 1 &\oplus X[4] \oplus X[1]X[2] \oplus X[1]X[3] \oplus X[1]X[4] \\ &\oplus X[1]X[2]X[3] \oplus X[1]X[2]X[4] \oplus X[1]X[3]X[4] \end{aligned} \tag{28}$$

TABLE II
EXAMPLE OF A 4-BIT S-BOX. EACH COORDINATE OF $s(X)$ CAN BE REPRESENTED BY A DEGREE-3 ANF REPRESENTATION

X	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$s(X)[1]$	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1
$s(X)[2]$	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0
$s(X)[3]$	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	1
$s(X)[4]$	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0

$$\begin{aligned} s(X)_3 = {} & 1 \oplus X[2] \oplus X[4] \oplus X[1]X[2] \oplus X[2]X[3] \\ & \oplus X[3]X[4] \oplus X[2]X[4] \oplus X[1]X[2]X[4] \\ & \oplus X[1]X[3]X[4] \qquad (29) \\ s(X)_4 = {} & 1 \oplus X[3] \oplus X[4] \oplus X[1]X[3] \oplus X[2]X[4] \\ & \oplus X[3]X[4] \oplus X[1]X[3]X[4] \oplus X[2]X[3]X[4]. \qquad (30) \end{aligned}$$

Desirably, S-box functions are designed such that the degree of the polynomial in S-box is large, which makes more difficult the application of higher order differential attacks. As the size of datasets grows, it is necessary to take advantage of the power of distributed computing, i.e., the data encryption computations are computed in a distributed manner. Let us consider the encryption problem of computing a 8-bit S-box function over a dataset $X = (X_1, \dots, X_{10})$ using a system of $N = 100$ workers. The best possible degree of 8-bit S-box $\deg s$ is equal to 7 [62]. For computing $s(X_1), \dots, s(X_{10})$ distributedly, the security threshold achieved by LCC is $\lfloor \frac{N-(K-1)\deg s-1}{2} \rfloor = 18$. Our proposed coded ANF and coded DNF provide the optimal security threshold of $\lfloor \frac{N-K}{2} \rfloor = 45$. As compared to LCC, the proposed coded ANF and coded DNF schemes improve the security threshold by 150%.

IX. EXTENSION TO GENERAL MULTIVARIATE POLYNOMIALS

In this section, we extend our problem to a more general computation model. More specifically, we focus on computing multivariate polynomial $f : \mathbb{V} \to \mathbb{U}$ over a dataset $X_1, \dots, X_K$ using a master and N workers, where $\mathbb{V}$ and $\mathbb{U}$ are arbitrary vector spaces over the certain field $\mathbb{F}$. We denote by $r(f)$ the number of monomials appearing in $f(X)$.[5]

As we see in the problem of computing Boolean functions, the security threshold provided by LCC can be low if $\deg f$ is high. To resolve such high degree difficulty which arises in computing general polynomials, we propose two different schemes: *coded data logarithm* and *coded data augmentation*. Especially, the proposed coded data logarithm scheme reduces the degree of polynomial computations by computing the logarithm of original data; and the proposed coded data augmentation reduces the degree of polynomial computations by pre-storing some low-degree monomials in advance.

A. Coded Data Logarithm

First, we illustrate the idea behind coded data logarithm by the following example.

[5]Similar to the case of Boolean functions, the total complexity of computing $f(X_1), \dots, f(X_K)$ is $O(Kr(f)\deg f)$.

Example 4: Consider the problem of computing function $f(X) = X^2$ in real field using 3 workers over a dataset $\vec{X} = (X_1, X_2)$, where input X_i's are 2×2 matrices.

We start by constructing a degree-1 multivariate polynomial for the function $f(X)$. The function $f(X) = X^2$ can be explicitly written as follows:

$$f(X) = \begin{bmatrix} [X]_{11}^2 + [X]_{12}[X]_{21} & [X]_{11}[X]_{12} + [X]_{12}[X]_{22} \\ {}_{11}[X]_{21} + [X]_{21}[X]_{22} & [X]_{12}[X]_{21} + [X]_{22}^2 \end{bmatrix}$$

which includes 7 monomials:

$$\begin{aligned} & [X]_{11}^2, \quad [X]_{22}^2, \quad [X]_{12}[X]_{21}, \quad [X]_{11}[X]_{12}, \\ & [X]_{12}[X]_{22}, \quad [X]_{11}[X]_{21}, \quad [X]_{21}[X]_{22}. \end{aligned}$$

By taking the logarithm of the absolute value of each monomial appearing in $f(X)$, we have

$$\begin{aligned} & 2\log|[X]_{11}|, \quad 2\log|[X]_{22}|, \quad \log|[X]_{12}| + \log|[X]_{21}|, \\ & \log|[X]_{11}| + \log|[X]_{12}|, \quad \log|[X]_{12}| + \log|[X]_{22}|, \\ & \log|[X]_{11}| + \log|[X]_{21}|, \quad \log|[X]_{21}| + \log|[X]_{22}|, \end{aligned}$$

which can be rewritten as:

$$\begin{aligned} & 2[W]_{11},\ 2[W]_{22},\ [W]_{12} + [W]_{21},\ [W]_{11} + [W]_{12}, \\ & [W]_{12} + [W]_{22}, [W]_{11} + [W]_{21},\ [W]_{21} + [W]_{22}, \end{aligned}$$

where $[W]_{ij} = \log|[X]_{ij}|$. We define a degree-1 multivariate polynomial $h(W)$ as follows:

$$\begin{aligned} h(W) = {} & [2[W]_{11},\ 2[W]_{22},\ [W]_{12} + [W]_{21},\ [W]_{11} + [W]_{12}, \\ & [W]_{12} + [W]_{22},\ [W]_{11} + [W]_{21},\ [W]_{21} + [W]_{22}]. \end{aligned}$$

To take advantage of the function $h(W)$ with the degree of 1, we take the logarithm of each entry's absolute value in X_1 and X_2 and define two matrices W_1 and W_2 as follows:

$$\begin{aligned} W_1 &= \begin{bmatrix} \log|[X_1]_{11}| & \log|[X_1]_{12}| \\ \log|[X_1]_{21}| & \log|[X_1]_{22}| \end{bmatrix}, \\ W_2 &= \begin{bmatrix} \log|[X_2]_{11}| & \log|[X_2]_{12}| \\ \log|[X_2]_{21}| & \log|[X_2]_{22}| \end{bmatrix}. \qquad (31) \end{aligned}$$

Then, we encode W_1 and W_2 to $\tilde{W}_1$, $\tilde{W}_2$ and $\tilde{W}_3$ using an $(3, 2)$ MDS code. Each worker n computes $h(\tilde{W}_n)$ where each entry of $h(W)$ is a linear combination of the logarithm of the corresponding X's entries' absolute values. By calculating the exponential of each entry in $h(W)$, the master can obtain the absolute values of all monomials appearing in $f(X)$, e.g., $[W]_{12} + [W]_{21} = \log|[X]_{12}| + \log|[X]_{21}| = \log|[X]_{12}[X]_{21}|$.

Computing the degree-1 (linear) function $h(W)$ allows us to apply a simple linear code to achieve the optimal security threshold.

In the following, we formally present the proposed coded data logarithm scheme. Given any multivariate polynomial function $f : \mathbb{V} \to \mathbb{U}$ over real field, the proposed coded data logarithm scheme first constructs the logarithmic data and a degree-1 multivariate polynomial function h by the followings.

1) *Logarithmic Data Construction:* For each X_k, we construct a logarithmic data W_k where each entry of W_k is the logarithm of X_k 's corresponding entry's absolute value,[6] i.e., $W_k[j] = \log |X_k[j]|$ where we denote by $X_k[j]$ the j-th input value of X_k without loss of generality.
2) *Degree-1 Multivariate Polynomial Construction:* Construct a multivariate polynomial function $h(W)$ with degree of 1 which computes the logarithm of absolute values of all monomials appearing in $f(X)$, i.e., for each monomial $\prod_{j\in\mathcal{S}} X[j]$ appearing in $f(X)$, the function $h(W)$ computes $\sum_{j\in\mathcal{S}} \log |X[j]| = \sum_{j\in\mathcal{S}} W[j]$.

After the construction of corresponding logarithmic data $W_1, \dots, W_K$ for $X_1, \dots, X_K$, the procedure of computations is as follows. The master encodes $W_1, \dots, W_K$ to $\tilde{W}_1, \dots, \tilde{W}_N$ using an (N, K) MDS code. Each worker n computes $h(\tilde{W}_n)$ and then sends the result back to the master. Upon receiving all results from the workers, the master first recovers $h(W_1), \dots, h(W_k)$ and calculates the exponential of each entry of $h(W_k)$ which recovers the absolute values of all monomials appearing in $f(X_k)$. Then, each monomial term can be determined by changing the sign accordingly. Lastly, the master recovers $f(X_1), \dots, f(X_K)$ by summing the monomial terms and the bias terms. Since each of $r(f)$ monomials has the degree up to $\deg f$, the complexity at each worker is $O((\deg f)r(f))$.

Reed-Solomon decoding is used for decoding the (N, K) MDS code. Successful decoding requires the number of errors of computation results such that $N \geq K + 2b$. The following theorem shows the security threshold achieved by the proposed coded data logarithm scheme.

Theorem 6: Given a number of workers N and a dataset $X = (X_1, \dots, X_K)$, the proposed coded data logarithm scheme can be robust to b adversaries for computing $\{f(X_k)\}_{k=1}^K$ for any multivariate polynomial f, as long as

$$N \geq K + 2b; \tag{32}$$

i.e., coded data logarithm achieves the security threshold

$$\beta_{\text{LOG}} = \left\lfloor \frac{N-K}{2} \right\rfloor. \tag{33}$$

Using a length-N Reed-Solomon code for each of $r(f)$ linear functions incurs the total complexity $O(r(f)N\log^2 N \log\log N)$. Computing the exponential of all the monomials incurs the complexity $O(Nr(f))$. Lastly, computing $f(X_1), \dots, f(X_K)$ by summing the monomials incurs the complexity $O(Nr(f))$. Thus, the total complexity of decoding step is $O(r(f)N\log^2 N \log\log N)$. Coded data logarithm provides the optimal security threshold, and has low decoding complexity for computing the sparse polynomials (small $r(f)$).

[6]Note that if there is any entry of $X_1, \dots, X_K$ is zero, we can replace that entry by a non-zero value and proceed the proposed scheme. Since the monomials with a zero entry is always equal to zero, we can set them to zero in the decoding process.

B. Coded Data Augmentation

In the following example, we show how the proposed coded data augmentation scheme reduces the degree of polynomial computations.

Example 5: Consider the problem of computing a multivariate polynomial function f with degree of 8 defined as follows:

$$f(X) = x_1^5 x_2^3 + x_2 x_3^3 + 2 \tag{34}$$

where each input X has three entries x_1, x_2, x_3.

To reduce the degree of computation such that using LCC can be robust to more adversaries in the system, we augment each input X by adding all degree-2 monomials as follows:

$$\begin{aligned}\bar{X} &= \left[x_1\ x_2\ x_3\ x_1^2\ x_2^2\ x_3^2\ x_1x_2\ x_1x_3\ x_2x_3\right] \\ &= \left[x_1\ x_2\ x_3\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\right].\end{aligned} \tag{35}$$

With data augmentation above, computing $f(X)$ is equivalent to computing $h(\bar{X})$ defined as follows:

$$h(\bar{X}) = y_1^2 y_2 y_4 + y_3 y_6 + 2 \tag{36}$$

which is the function with degree of 4.

By prestoring the twice amount of data in each worker, the system can be robust to number of $\frac{4(K-1)}{2} = 2(k-1)$ more adversaries using LCC. Such pre-storing some low-degree polynomials enable us to enhance the robustness against Byzantine workers in the system.

In the following, we formally present the proposed coded data augmentation scheme. Given any multivariate polynomial $f : \mathbb{V} \to \mathbb{U}$ over a field $\mathbb{F}$ with an integer q, coded data augmentation first augments data and construct a low degree polynomial as follows.

1) *Data Augmentation:* For each X_k, we construct $\bar{X}_k$ by adding all the monomials of X_k's entries with the degree up to q, i.e., adding $\prod_{j\in\mathcal{S}} X[j]$ for all $\mathcal{S} \subseteq [q]$.
2) *Low Degree Polynomial Construction:* By substituting each added monomial as a new variable, we construct a multivariate polynomial function $h(\bar{X}_k)$ with degree of $u + \mathbb{1}_{\{r>0\}}$, in which degree of f can be uniquely written as $\deg f = qu + r$ and $0 \leq r \leq q - 1$. We note that such constructed polynomial is not unique but degree of h is unique.

The procedure of computations is as follows. The master encodes $\bar{X}_1, \dots, \bar{X}_K$ to $\tilde{X}_1, \dots, \tilde{X}_N$ using LCC encoder. Each worker n computes $h(\tilde{X}_n)$ and then sends the result back to the master. Whenever the master receives N results from the workers, the master recovers $h(\bar{X}_1), \dots, h(\bar{X}_K)$ using a length-N Reed-Solomon code. Lastly, the master has $f(X_1) = h(\bar{X}_1), f(X_2) = h(\bar{X}_2), \dots, f(X_K) = h(\bar{X}_K)$. Since each of $r(f)$ monomials has the degree up to $u + \mathbb{1}_{\{r>0\}}$, the complexity at each worker is $O((u + \mathbb{1}_{\{r>0\}})r(f))$.

Because the constructed function $h(\bar{X})$ has degree of $u + \mathbb{1}_{\{r>0\}}$ ($\deg f = qu + r$), to be robust to b adversaries, LCC requires the number of workers N such that $N \geq (K-1)(u + \mathbb{1}_{\{r>0\}}) + 2b + 1$. Then, we have the following theorem.

Theorem 7: Given a number of workers N and a dataset $X = (X_1, \ldots, X_K)$, the proposed coded data augmentation scheme with parameter q can be robust to b adversaries for computing $\{f(X_k)\}_{k=1}^{K}$ for any multivariate polynomial f, as long as

$$N \geq (K-1)\big(u + \mathbb{1}_{\{r>0\}}\big) + 2b + 1; \tag{37}$$

i.e., coded data augmentation with parameter q achieves the security threshold

$$\beta_{\text{AUG}} = \left\lfloor \frac{N - (K-1)\big(u + \mathbb{1}_{\{r>0\}}\big) - 1}{2} \right\rfloor. \tag{38}$$

where $\deg f = qu + r$ and $0 \leq r \leq q - 1$.

Decoding the computation results using a length-N Reed-Solomon code for the constructed polynomial function incurs the total complexity $O(N \log^2 N \log\log N)$. By trading the cost of storing more data for improving robustness against adversarial workers, coded data augmentation can be applied to any multivariate general polynomials and robust to $(K-1)(\deg f - u - \mathbb{1}_{\{r>0\}})/2$ more adversaries than LCC.

Remark 8: Since computing all K evaluations $f(X_1), \ldots, f(X_K)$ at the master incurs the total complexity $O(Kr(f)\deg f)$, it is more efficient to use coded data logarithm when $\deg f > \frac{N}{K} \log^2 N \log\log N$. When $r(f)\deg f > \frac{N}{K} \log^2 N \log\log N$, coded data augmentation is more efficient than computing all the evaluations at the master.

X. Concluding Remarks and Future Directions

In this article, we focus on computing a Boolean function in a distributed manner against adversarial servers. To resolve the degree problem of using LCC (i.e., the security threshold provided by LCC can be low if the polynomial's degree is high), the proposed schemes called coded ANF, coded DNF and coded PTF largely improve the security threshold by modeling the polynomial as the concatenation of some low-degree polynomial functions and threshold functions. It is shown that coded ANF and coded DNF are optimal by matching to the derived theoretical outer bound; and increase the security threshold by 150% for computing 8-bit S-box in the application of block cyphers using a distributed computing system with 100 workers.

There are many interesting directions can be pursued on the problem of coded Boolean computations. For example, the proposed coded ANF and coded DNF require embedding bits to reals, which might lead to some floating-point errors during decoding process. Thus, one direction is to implement two schemes in an actual computing system and measure the effect of field transformation.

Acknowledgment

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.

References

[1] C.-S. Yang and A. S. Avestimehr, "Coded computing for Boolean functions," 2020. [Online]. Available: arXiv:2001.08720.

[2] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[3] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2017, pp. 119–129.

[4] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation*. Cambridge, U.K.: Cambridge Univ. Press, 2015.

[5] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proc. Eur. Symp. Res. Comput. Security*, 2008, pp. 192–206.

[6] T. W. Cusick and P. Stanica, *Cryptographic Boolean Functions and Applications*. London, U.K.: Academic, 2017.

[7] S. Li, M. Yu, C. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, "PolyShard: Coded sharding achieves linearly scaling efficiency and security simultaneously," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 249–261, 2020.

[8] S. Cao, S. Kadhe, and K. Ramchandran, "CoVer: Collaborative light-node-only verification and data availability for blockchains," 2020. [Online]. Available: arXiv:2010.00217

[9] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 17–30.

[10] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. 22nd Int. Conf. Artif. Intell. Stat.*, 2019, pp. 1215–1225.

[11] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

[12] H.-D. Block, "The perceptron: A model for brain functioning. I," *Rev. Mod. Phys.*, vol. 34, no. 1, p. 123, 1962.

[13] M. Minsky, *Perceptrons*. Cambridge, MA, USA: MIT Press, 1969.

[14] N. Nisan and M. Szegedy, "On the degree of Boolean functions as real polynomials," *Comput. Complexity*, vol. 4, no. 4, pp. 301–313, 1994.

[15] J. Aspnes, R. Beigel, M. Furst, and S. Rudich, "The expressive power of voting polynomials," *Combinatorica*, vol. 14, no. 2, pp. 135–148, 1994.

[16] R. O'Donnell and R. A. Servedio, "New degree bounds for polynomial threshold functions," in *Proc. 35th Annu. ACM Symp. Theory Comput.*, 2003, pp. 325–334.

[17] R. O'Donnell and R. A. Servedio, "Extremal properties of polynomial threshold functions," *J. Comput. Syst. Sci.*, vol. 74, no. 3, pp. 298–312, 2008.

[18] M. Bun and J. Thaler, "A nearly optimal lower bound on the approximate degree of AC ^0," *SIAM J. Comput.*, vol. 49, no. 4, pp. 59–96, 2019.

[19] E. Oztop, "An upper bound on the minimum number of monomials required to separate dichotomies of $\{-1, 1\}$ n," *Neural Comput.*, vol. 18, no. 12, pp. 3119–3138, 2006.

[20] K. Amano, "New upper bounds on the average PTF density of Boolean functions," in *Proc. Int. Symp. Algorithms Comput.*, 2010, pp. 304–315.

[21] C. E. Sezener and E. Oztop, "Heuristic algorithms for obtaining polynomial threshold functions with low densities," 2015. [Online]. Available: arXiv:1504.01167

[22] A. R. Klivans, R. O'Donnell, and R. A. Servedio, "Learning intersections and thresholds of halfspaces," *J. Comput. Syst. Sci.*, vol. 68, no. 4, pp. 808–840, 2004.

[23] J. Alman, T. M. Chan, and R. Williams, "Polynomial representations of threshold functions and algorithmic applications," in *Proc. IEEE 57th Annu. Symp. Found. Comput. Sci. (FOCS)*, New Brunswick, NJ, USA, 2016, pp. 467–476.

[24] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

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

[26] 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*. Red Hook, NY, USA: Curran, 2016, pp. 2100–2108.

[27] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, 2017, pp. 2418–2422.
[28] Q. Yu, M. A. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2017, pp. 4403–4413.
[29] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 3368–3376.
[30] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 34–40, Apr. 2017.
[31] K. G. Narra, Z. Lin, M. Kiamari, S. Avestimehr, and M. Annavaram, "Slack squeeze coded computing for adaptive straggler mitigation," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2019, p. 14.
[32] S. Li and S. Avestimehr, "Coded computing," in *Foundations and Trends® in Communications and Information Theory*, vol. 17, no. 1, pp. 1–148, 2020.
[33] S. Prakash, A. Reisizadeh, R. Pedarsani, and A. S. Avestimehr, "Coded computing for distributed graph analytics," *IEEE Trans. Inf. Theory*, vol. 66, no. 10, pp. 6534–6554, Oct. 2020.
[34] B. Güler, A. S. Avestimehr, and A. Ortega, "TACC: Topology-aware coded computing for distributed graph processing," *IEEE Trans. Signal Inf. Process. over Netw.*, vol. 6, pp. 508–525, May 2020, doi: 10.1109/TSIPN.2020.2998223.
[35] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1920–1933, Mar. 2020.
[36] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.
[37] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, USA, 2018, pp. 1620–1624.
[38] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "DRACO: Byzantine-resilient distributed training via redundant gradients," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 903–912.
[39] S. Kadhe, O. O. Koyluoglu, and K. Ramchandran, "Gradient coding based on block designs for mitigating adversarial stragglers," 2019. [Online]. Available: arXiv:1904.13373
[40] H. A. Nodehi and M. A. Maddah-Ali, "Secure coded multi-party computation for massive matrix operations," 2019. [Online]. Available: arXiv:1908.04255
[41] J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, "CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning," 2019. [Online]. Available: arXiv:1902.00641
[42] J. So, B. Guler, and A. S. Avestimehr, "A scalable approach for privacy-preserving collaborative machine learning," 2020. [Online]. Available: arXiv:2011.01963
[43] Q. Yu and A. S. Avestimehr, "Coded computing for resilient, secure, and privacy-preserving distributed matrix multiplication," *IEEE Trans. Commun.*, vol. 69, no. 1, pp. 59–72, Jan. 2021.
[44] M. Soleymani, H. Mahdavifar, and A. S. Avestimehr, "Analog lagrange coded computing," *IEEE J. Sel. Areas Inf. Theory*, early access, 2021, doi: 10.1109/JSAIT.2021.3056377
[45] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2017, pp. 5434–5442.
[46] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, early access, Jan. 26, 2021, doi: 10.1109/JSAIT.2021.3054610.
[47] S. Prakash, A. Reisizadeh, R. Pedarsani, and A. S. Avestimehr, "Hierarchical coded gradient aggregation for learning at the edge," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Los Angeles, CA, USA, 2020, pp. 2616–2621.
[48] S. Prakash *et al.*, "Coded computing for low-latency federated learning over wireless edge networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 233–250, Jan. 2021.
[49] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, "Coded merkle tree: Solving data availability attacks in blockchains," in *Proc. Int. Conf. Financ. Cryptogr. Data Security*, 2020, pp. 114–134.
[50] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely coded computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2019, pp. 2798–2802.
[51] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2019, pp. 301–310.
[52] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Edge computing in the dark: Leveraging contextual-combinatorial bandit and coded computing," 2019. [Online]. Available: arXiv:1912.09512.
[53] R. O'Donnell, *Analysis of Boolean Functions*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[54] E. Berlekamp, "Nonbinary BCH decoding (abstr.)," *IEEE Trans. Inf. Theory*, vol. 14, no. 2, p. 242, Mar. 1968.
[55] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory*, vol. 15, no. 1, pp. 122–127, Jan. 1969.
[56] A. R. Klivans and R. A. Servedio, "Learning DNF in time $2^{O(n1/3)}$," *J. Comput. Syst. Sci.*, vol. 68, no. 2, pp. 303–318, 2004.
[57] E. Kushilevitz, "A simple algorithm for learning O(log n)-term DNF," *Inf. Process. Lett.*, vol. 61, no. 6, pp. 289–292, 1997.
[58] A. Blum, "Rank-r decision trees are a subclass of r-decision lists," *Inf. Process. Lett.*, vol. 42, no. 4, pp. 183–185, 1992.
[59] C. Carlet, "Boolean functions for cryptography and error-correcting codes," in *Boolean Models and Methods in Mathematics, Computer Science, and Engineering* (Encyclopedia of Mathematics and its Applications), Y. Crama and L. P. Hammer, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2010, pp. 257–397.
[60] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Proc. Workshop Theory Appl. Cryptograph. Techn.*, 1993, pp. 386–397.
[61] S. Bajrić, "Implementing symmetric cryptography using sequence of semi-bent functions," in *Modern Cryptography*, M. Domb, Ed. Rijeka, Croatia: IntechOpen, 2019, ch. 1. [Online]. Available: https://doi.org/10.5772/intechopen.85023
[62] E. Boss, V. Grosso, T. Güneysu, G. Leander, A. Moradi, and T. Schneider, "Strong 8-bit sboxes with efficient masking in hardware," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.*, 2016, pp. 171–193.

Chien-Sheng Yang (Graduate Student Member, IEEE) received the B.S. degree in electrical and computer engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2015. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Southern California, Los Angeles. His interests include information theory, machine learning, and edge computing. He received the Annenberg Graduate Fellowship in 2016. He was a finalist of the ACM International Symposium on Mobile Ad Hoc Networking and Computing Best Paper Award in 2019.

A. Salman Avestimehr (Fellow, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology in 2003, and the M.S. degree in electrical engineering and computer science and the Ph.D. degree from the University of California, Berkeley, in 2005 and 2008, respectively.

He is a Professor and the Director of the Information Theory and Machine Learning (vITAL) Research Laboratory, Electrical and Computer Engineering Department, University of Southern California. He is also an Amazon Scholar with Alexa AI. His research interests include information theory and coding theory, and large-scale distributed computing and machine learning, secure and private computing, and blockchain systems. He has received a number of awards for his research, including the James L. Massey Research & Teaching Award from IEEE Information Theory Society, the Information Theory Society and Communication Society Joint Paper Award, the Presidential Early Career Award for Scientists and Engineers from the White House, the Young Investigator Program Award from the U.S. Air Force Office of Scientific Research, the National Science Foundation CAREER Award, the David J. Sakrison Memorial Prize, and several best paper awards at conferences. He is currently the General Co-Chair of the 2020 International Symposium on Information Theory. He has been an Associate Editor for IEEE TRANSACTIONS ON INFORMATION THEORY.