PRAC: Private and Rateless Adaptive Coded Computation at the Edge

Rawad Bitar*, Yuxuan Xing†, Yasaman Keshtkarjahromi‡, Venkat Dasari§,
Salim El Rouayheb*, and Hulya Seferoglu†

*Rutgers the State University of New Jersey,
†University of Illinois at Chicago,
‡Seagate Technology, R&D,
§US Army Research Lab

Abstract

Edge computing is emerging as a new paradigm to allow processing data near the edge of the network, where the data is typically generated and collected. This enables critical computations at the tactical edge in applications such as Internet of Battlefield Things (IoBT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints.

Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation. Coded computation is recently gaining interest, thanks to its higher reliability, smaller delay, and lower communication costs. In this paper, we develop a private and rateless adaptive coded computation (PRAC) algorithm by taking into account (i) the privacy requirements of IoBT applications and devices, and (ii) the heterogeneous and time-varying resources of edge devices. We show that PRAC outperforms known secure coded computing methods when resources are heterogeneous. We provide theoretical guarantees on the performance of PRAC and its comparison to baselines. Moreover, we confirm our theoretical results through simulations.

I. Introduction

Edge computing is emerging as a new paradigm to allow processing data near the edge of the network, where the data is typically generated and collected. This enables critical computation at the tactical edge in applications such as Internet of Battlefield Things (IoBT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints.

One of the promising solutions to handle computationally intensive tasks is computation offloading, which advocates offloading tasks to remote servers or cloud. Yet, offloading tasks to remote servers or cloud could be a luxury that cannot be afforded by most of the edge applications, where connectivity to remote servers can be lost or compromised, which makes edge computing crucial. Connectivity constraints are more prominent when dealing with tactical edge computing where the network connectivity is adversely affected by resources as well as hostile network elements.

Edge computing advocates that computationally intensive tasks in a device (master) could be offloaded to other edge or end devices (workers) in close proximity. However, offloading tasks to other devices leaves the IoBT and the applications it is supporting at the complete mercy of an attacker. Furthermore, exploiting the potential of edge computing is challenging mainly due to the heterogeneous and time-varying nature of the devices at the edge. Thus, our goal is to develop a private, dynamic, adaptive, and heterogeneity-aware cooperative computation framework that provides both privacy and computation efficiency guarantees.

Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation [1]. The following canonical example demonstrates the effectiveness of coded computation.

Example 1. *Consider the setup where a master device wishes to offload a task to 3 workers. The master has a large data matrix A and wants to compute matrix vector product $A\mathbf{x}$. The master device divides the matrix A row-wise equally into two smaller matrices A_1 and A_2, which are then encoded using a $(3,2)$ Maximum Distance Separable (MDS) code*[1] *to give $B_1 = A_1$, $B_2 = A_2$ and $B_3 = A_1 + A_2$, and sends each to a different worker. Also, the master device sends $\mathbf{x}$*

[1]An (n,k) MDS code divides the master's data into k chunks and encodes it into n chunks ($n > k$) such that any k chunks out of n are sufficient to recover the original data.

TABLE I: Example PRAC operation in homogeneous and static setup.

Time	Worker 1	Worker 2	Worker 3
1	R_1	$A_1 + A_3 + R_1$	$A_3 + R_1$
2		R_2	
3	$A_2 + A_3 + R_2$		
4			$A_2 + R_2$

to workers and ask them to compute $B_i\mathbf{x}$, $i \in \{1, 2, 3\}$. When the master receives the computed values (i.e., $B_i\mathbf{x}$) from at least two out of three workers, it can decode its desired task, which is the computation of $A\mathbf{x}$. The power of coded computations is that it makes $B_3 = A_1 + A_2$ acts as a "joker" redundant task that can replace any of the other two tasks if they end up straggling or failing. □

The above toy example demonstrates the benefit of coding for edge computing. However, the very nature of task offloading from a master to worker devices makes the computation framework vulnerable to attacks. One of the attacks, which is also the focus of this work, is *eavesdropper adversary*, where one or more of the workers can behave as an eavesdropper and can spy on the coded data sent to these devices for computations.[2] For instance, $B_3 = A_1 + A_2$ in Example 1 can be processed and spied on by worker 3. Even though $A_1 + A_2$ is coded, the attacker can infer some information from this coded task. Thus, it is crucial to develop a private coded computation algorithm against eavesdropper adversary.

In this paper, we develop a private and rateless adaptive coded computation (PRAC) algorithm. PRAC is (i) private as it is secure against eavesdropper adversary, (ii) rateless, because it uses Fountain codes [2]–[4] instead of Maximum Distance Separable (MDS) codes [5], [6], and (iii) adaptive as the master device offloads tasks to workers by taking into account their heterogeneous and time-varying resources. Next, we illustrate the main idea of PRAC through an illustrative example.

Example 2. *We consider the same setup in Example 1, where a master device offloads a task to 3 workers. The master has a large data matrix A and wants to compute matrix vector product $A\mathbf{x}$. The master device divides matrix A row-wise into 3 sub-matrices A_1, A_2, A_3; and encodes these matrices using a Fountain code[3] [2]–[4]. An example set of coded packets is A_2, A_3, $A_1 + A_3$, and $A_2 + A_3$. However, prior to sending a coded packet to a worker, the master generates a random key matrix R with the same dimensions as A_i and with entries drawn uniformly from the same field which contains the entries of A. The key matrix is added to the coded packets to provide privacy as shown in Table I. In particular, a key matrix R_1 is created at the start of time slot 1, combined with $A_1 + A_3$ and A_3, and transmitted to workers 2 and 3, respectively. R_1 is also transmitted to worker 1 in order to obtain $R_1\mathbf{x}$ that will help the master in the decoding process. The computation of $(A_1 + A_3 + R_1)\mathbf{x}$ is completed at the end of time slot 2. Thus, at that time slot the master generates a new matrix. R_2 and sends it worker 2. At time slot 3, worker 1 finishes its computation, therefore the master adds R_2 to $A_2 + A_3$ and sends it to worker 1. A similar process is repeated at time slot 4. Now the master waits for worker 2 to return $R_2\mathbf{x}$ and for any other worker to return its uncompleted task in order to decode $A\mathbf{x}$. Thanks to using key matrices R_1 and R_2, and assuming that workers do not collude, privacy is guaranteed.* □

This example shows that PRAC can take advantage of coding for computation, and provide privacy.

Contributions. We design PRAC for heterogeneous and time-varying private coded computing with colluding workers. In particular, PRAC encodes sub-tasks using Fountain codes, and determines how many coded packets and keys each worker should compute dynamically over time. We provide theoretical analysis of PRAC and show that (i) it guarantees privacy conditions, and (ii) for a given number of transmitted packets, PRAC uses minimum number of random matrices to satisfy privacy requirements. Furthermore, we provide a closed form task completion delay analysis of PRAC. Finally, we evaluate the performance of PRAC via simulations.

The structure of the rest of this paper is as follows. We start with presenting the system model in Section II. In Section III we present the design of private and rateless adaptive coded computation (PRAC). We present evaluation results in section IV. We present the related work in Section V presents related work and conclude the paper in Section VI.

[2]Note that this work focuses specifically on *eavesdropper adversary* although there are other types of attacks; for example *Byzantine adversary*, which is out of scope of this work.

[3]Fountain codes are desirable here for two properties: (i) they provide a fluid abstraction of the coded packets so the master can always decode with high probability as long as it collects enough packets; (ii) They have low decoding complexity.

II. System Model

Setup. We consider a master/workers setup at the edge of the network, where the master device M offloads its computationally intensive tasks to workers $w_i,\ i \in \mathcal{N}$, (where $|\mathcal{N}| = n$) via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. The master device divides a task into smaller sub-tasks, and offloads them to workers that process these sub-tasks in parallel.

Task Model. We focus on the computation of linear functions, i.e., matrix-vector multiplication. We suppose the master wants to compute the matrix vector product $A\mathbf{x}$, where $A \in \mathbb{F}_q^{m\times\ell}$ can be thought of as the data matrix and $\mathbf{x} \in \mathbb{F}_q^{\ell}$ can be thought of as an attribute vector. We assume that the entries of A and $\mathbf{x}$ are drawn independently and uniformly at random[4] from $\mathbb{F}_q$. The motivation stems from machine learning applications where computing linear functions is a building block of several iterative algorithms such as gradient descent. For instance, the main computation of a gradient descent algorithm with squared error loss function is

$$\mathbf{x}^+ = \mathbf{x} - \alpha A^T(A\mathbf{x} - \mathbf{y}), \tag{1}$$

where $\mathbf{x}$ is the value of the attribute vector at a given iteration, $\mathbf{x}^+$ is the updated value of $\mathbf{x}$ at this iteration and the learning rate α is a parameter of the algorithm. Equation (1) consists of computing two linear functions $A\mathbf{x}$ and $A^T\mathbf{w} \triangleq A^T(A\mathbf{x} - \mathbf{y})$.

Worker and Attack Model. The workers incur random delays while executing the task assigned to them by the master device. The workers have different computation and communication specifications resulting in a heterogeneous environment which includes workers that are significantly slower than others, known as stragglers. Moreover, the workers cannot be trusted with the master's data. We consider an *eavesdropper adversary* in this paper, where one or more of workers can be eavesdroppers and can spy on the coded data sent to these devices for computations. We assume that up to z, $z < n$, workers can collude, i.e., z workers can share the data they received from the master in order to obtain information about A. The parameter z can be chosen based on the desired security level; a larger z means a higher security level and vice versa. One would want to set z to the largest possible value for maximum, $z = n - 1$ security purposes. However, this has the drawback of increasing the complexity and the runtime of the algorithm. In our setup we assume that z is a fixed and given system parameter.

Coding & Secret Keys. The matrix A can be divided into b row blocks (we assume that b divides m, otherwise all-zero rows can be added to the matrix to satisfy this property) denoted by $A_i,\ i = 1\ldots, m/b$. The master applies Fountain coding [2]–[4] across row blocks to create information packets $\nu_j \triangleq \sum_{i=1}^{m} c_{i,j}A_i,\ j = 1, 2, \ldots,$ where the $c_{i,j} \in \{0, 1\}$. Note that an information packet is a matrix of dimension $m/b \times \ell$, i.e., $\nu_j \in \mathbb{F}_q^{m/b\times\ell}$. Such rateless coding is compatible with our goal to create adaptive coded cooperation computation framework. In order to maintain privacy of the data, the master device generates random matrices R_i of dimension $m/b \times \ell$ called *keys*. The entries of the R_i matrices are drawn uniformly at random from the field that contains the entries of A. Each information packet ν_j is *padded* with a linear combination of z keys $f_j(R_{i_1}, \ldots, R_{i_z})$ to create a secure packet $s_j \in \mathbb{F}_q^{m/b\times\ell}$ defined as $s_j \triangleq \nu_j + f_j(R_{i_1}, \ldots, R_{i_z})$.

The master device sends $\mathbf{x}$ to all workers, then it sends the keys and the s_j's to the workers according to our PRAC scheme described later. Each worker multiplies the received packet by $\mathbf{x}$ and sends the result back to the master. Since the encoding is rateless, the master keeps sending packets to the workers until it can decode $A\mathbf{x}$. The master then sends a stop message to all the workers.

Privacy Conditions. Our primary requirement is that any collection of z (or less) workers will not be able to obtain any information about A, in an information theoretic sense.

In particular, let $P_i,\ i = 1\ldots, n$, denote the collection of packets sent to worker w_i. For any set $\mathcal{B} \subseteq \{1, \ldots, n\}$, let $P_{\mathcal{B}} \triangleq \{P_i, i \in \mathcal{B}\}$ denote the collection of packets given to worker w_i for all $i \in \mathcal{B}$. The privacy requirement[5] can be expressed as

$$H(A|P_{\mathcal{Z}}) = H(A), \quad \forall \mathcal{Z} \subseteq \{1, \ldots, n\} \text{ s.t. } |\mathcal{Z}| \leq z. \tag{2}$$

$H(A)$ denotes the entropy, or uncertainty, about A and $H(A|P_{\mathcal{Z}})$ denotes the uncertainty about A after observing $P_{\mathcal{Z}}$.

Delay Model. Each packet transmitted from the master to a worker $w_i,\ i = 1, 2, ..., n$, experiences the following delays: (i) transmission delay for sending the packet from the master to the worker, (ii) computation delay for computing the

[4]We abuse notation and denote both the random matrix representing the data and its realization by A. We do the same for $\mathbf{x}$.

[5]In some cases the vector $\mathbf{x}$ may contain information about A and therefore must not be revealed to the workers. We explain in the technical report [7, Appendix A] how to generalize our scheme to account for such cases.

TABLE II: Summary of notations.

Symbol	Meaning
M	master
w_i	worker i
n	number of workers
A	$m \times \ell$ data matrix
$\mathbf{x}$	$1 \times \ell$ attribute vector
z	number of colluding workers
m	number of rows in A
ε	overhead of Fountain codes
A_i	i^{th} row block of data matrix A

Symbol	Meaning
R	random matrix
RTT_i	round trip time to send and receive packet i
$\beta_{t,i}$	computation time of the t^{th} packet at w_i
ν	Fountain coded packet of A_i's
s	secure Fountain coded packet
T_i	time to compute a packet at w_i
$T_{(d)}$	d^{th} order statistic of T_i's
T	time spent by M to decode $A\mathbf{x}$

TABLE III: Depiction of PRAC in the presence of stragglers. The master keeps generating packets using Fountain codes until it can decode $A\mathbf{x}$. The master estimates the average task completion time of each worker and sends a new packet to avoid idle time. Each new packet sent to a worker must be secured with a new random vector. The master can decode $A_1\mathbf{x}, \ldots, A_6\mathbf{x}$ after receiving all the packets not having $R_{4,1}$ or $R_{4,2}$ in them.

Time	Worker 1	Worker 2	Worker 3	Worker 4
1	$R_{1,1}$	$R_{1,2}$	$A_4 + R_{1,1} + R_{1,2}$	$A_3 + A_4 + A_6 + R_{1,1} + 2R_{1,2}$
2				$R_{2,1}$
3	$R_{2,2}$			
4		$A_3 + R_{2,1} + R_{2,2}$	$A_4 + A_5 + R_{2,1} + 2R_{2,2}$	
5		$R_{3,1}$		
6	$A_2 + R_{3,1} + R_{3,2}$			$R_{3,2}$
7		$R_{4,1}$	$A_1 + R_{3,1} + 2R_{3,2}$	
8	$R_{4,2}$			$A_2 + A_3 + R_{4,1} + R_{4,2}$

multiplication of the packet by the vector $\mathbf{x}$, and (iii) transmission delay for sending the computed packet from the worker w_i back to the master. We denote by $\beta_{t,i}$ the computation time of the t^{th} packet at worker i and RTT_i denotes the round-trip time spent to send and receive a packet from worker i. The time spent by the master is equal to the time taken by the $(z+1)^{\text{st}}$ fastest worker to finish its assigned tasks.

III. Design of PRAC

A. Overview

We present the detailed explanation of PRAC. Let $p_{t,i} \in \mathbb{F}_q^{m/b \times \ell}$ be the t^{th} packet sent to worker w_i. This packet can be either a key or a secure packet. For each value of t, the master sends z keys denoted by $R_{t,1}, \ldots, R_{t,z}$ to z different workers and up to $n-z$ secure packets $s_{t,1}, \ldots, s_{t,n-z}$ to the remaining workers. The master needs the results of $m+\epsilon$ information packets, i.e., $\nu_{t,i}\mathbf{x}$, to decode the final result $A\mathbf{x}$, where ϵ is the overhead required by Fountain coding[6]. To obtain the results of $m+\epsilon$ information packets, the master needs the results of $m+\epsilon$ secure packets, $s_{t,i}\mathbf{x} = (\nu_{i,j} + f_j(R_{t,i}, \ldots, R_{t,z}))\mathbf{x}$, together with the all the corresponding[7] $R_{t,i}\mathbf{x}, i = 1, \ldots, z$. Therefore, only the results of the $s_{t,i}\mathbf{x}$ for which all the computed keys $R_{t,i}\mathbf{x}, i = 1, ..., z$, are received by the master can account for the total of $m+\epsilon$ information packets.

B. Dynamic Rate Adaptation

The dynamic rate adaptation part of PRAC is based on [8]. In particular, the master offloads coded packets gradually to workers and receives two ACKs for each transmitted packet; one confirming the receipt of the packet by the worker, and the second one (piggybacked to the computed packet) showing that the packet is computed by the worker. Then, based on the frequency of the received ACKs, the master decides to transmit more/less coded packets to that worker. In particular, each packet $p_{t,i}$ is transmitted to each worker w_i before or right after the computed packet $p_{t-1,i}\mathbf{x}$ is received at the master. For this purpose, the average per packet computing time $\mathbb{E}[\beta_{t,i}]$ is calculated for each worker w_i dynamically based on the previously received ACKs. Each packet $p_{t,i}$ is transmitted after waiting $\mathbb{E}[\beta_{t,i}]$ from the time

[6]The overhead required by Fountain coding is typically as low as 5% [4], i.e., $\epsilon = 0.05m$.

[7]Recall that $f_j(R_{t,i}, \ldots, R_{t,z})$ is a linear function, thus it is easy to extract $(f_j(R_{t,i}, \ldots, R_{t,z}))\mathbf{x}$ from $(R_{t,i})\mathbf{x}, i = 1, ..., z$.

$p_{t-1,i}$ is sent or right after packet $p_{t-1,i}\mathbf{x}$ is received at the master, thus reducing the idle time at the workers. This policy is shown to approach the optimal task completion delay and maximizes the workers' efficiency and is shown to improve task completion time significantly compared with the literature [8].

C. Coding

We explain the coding scheme used in PRAC. We start with an example to build an intuition and illustrate the scheme before going into details.

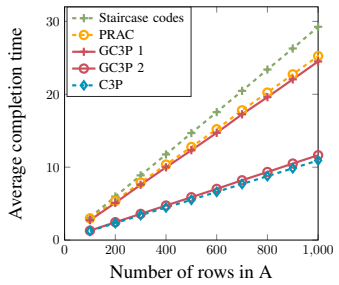
Example 3. *Assume there are $n = 4$ workers out of which any $z = 2$ can collude. Let A and $\mathbf{x}$ be the data owned by the master and the vector to be multiplied by A, respectively. The master sends $\mathbf{x}$ to all the workers. For the sake of simplicity, assume A can be divided into $b = 6$ row blocks, i.e., $A = \begin{bmatrix} A_1 & A_2 & \dots & A_6 \end{bmatrix}^T$. The master encodes the A_i's using Fountain code. We denote by* round *the event when the master sends a new packet to a worker. For example, we say that worker 1 is at round 3 if it has received 3 packets so far. For every round t, the master generates $z = 2$ random matrices $R_{t,1},\ R_{t,2}$ and encodes them using an $(n, z) = (4, 2)$ systematic maximum distance separable (MDS) code by multiplying $R_{t,1},\ R_{t,2}$ by a generator matrix G as follows*

$$G \begin{bmatrix} R_{t,1} \\ R_{t,2} \end{bmatrix} \triangleq \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} R_{t,1} \\ R_{t,2} \end{bmatrix}. \tag{3}$$

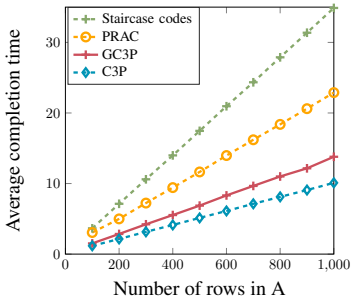
The following ($R_{t,1},\ R_{t,2},\ R_{t,1} + R_{t,2},\ R_{t,1} + 2R_{t,2}$) is the resulting encoding of the $R_{t,i}$'s. Now let us assume that workers can be stragglers. At the beginning the master initializes all the workers at round 1. Afterwards, when a worker w_i finishes its task, the master checks how many packets this worker has received so far and how many other workers are at this round. If this worker w_i is the first or second to be at round t, the master generates $R_{t,1}$ or $R_{t,2}$, respectively, and sends it to w_i. Otherwise, if w_i is the j^{th} worker ($j > 2$) to be at round t, the master multiplies $\begin{bmatrix} R_{t,1} & R_{t,2} \end{bmatrix}^T$ by the j^{th} row of G, adds it to a generated coded packet, and sends it to w_i. The master keeps sending packets to the workers until it can decode $A\mathbf{x}$. We illustrate the idea in Table III.

We now explain the details of PRAC in the presence of z colluding workers.

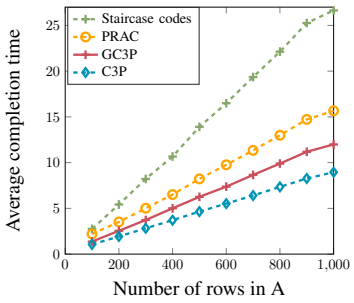
1) *Initialization:* The master divides A into b row blocks $A_1, \dots, A_b$ and sends the vector $\mathbf{x}$ to the workers. Let $G \in \mathbb{F}_q^{n \times z}$, $q > n$, be the generator matrix of an (n, z) systematic MDS code. For example one may use systematic Reed-Solomon codes that use Vandermonde matrix as generator matrix, see for example [9]. The master generates z random matrices $R_{1,1}, \dots, R_{1,z}$ and encodes them using G. Each coded key can be denoted by $\mathbf{g}_i\mathcal{R}$ where $\mathbf{g}_i$ is the i^{th} row of G and $\mathcal{R} \triangleq \begin{bmatrix} R_{1,1} & \dots & R_{1,z} \end{bmatrix}^T$. The master sends the z keys $R_{1,1}, \dots, R_{1,z}$ to the first z workers, generates $n - z$ Fountain coded packets of the A_i's, adds to each packet an encoded random key $\mathbf{g}_i\mathcal{R}$, $i = z + 1, \dots n$, and sends them to the remaining $n - z$ workers.
2) *Encoding and adaptivity:* When the master wants to send a new packet to a worker (noting that a packet $p_{t,i}$ is transmitted to worker w_i before, or right after, the computed packet $p_{t-1,i}\mathbf{x}$ is received at the master according to the strategy described in Section III-B), it checks at which round this worker is, i.e., how many packets this worker has received so far, and checks how many other workers are at this round. Assume the worker is at round t and $j - 1$ other workers are at this round. If $j \leq z$, the master generates and sends $R_{t,j}$ to the worker. However, if $j > z$ the master generates a Fountain coded packet of the A_i's (e.g., $A_1 + A_2$), adds to it $\mathbf{g}_j\mathcal{R}$ and sends the packet $(A_1 + A_2 + \mathbf{g}_j\mathcal{R})$ to the worker. Each worker computes the multiplication of the received packet by the vector $\mathbf{x}$ and sends the result to the master.
3) *Decoding and speed:* Let τ_i denote the number of packets received by worker i. At the end of the process, the master has $R_{t,i}\mathbf{x}$ for all $t = 1, \dots, \tau_{max}$ and all $i = 1, \dots, z$, where $\tau_{max} \triangleq \max_i \tau_i$. The master can subtract the $R_{t,i}$'s from all received secure information packets, thus can decode the A_i's using the Fountain code decoding process. The number of secure packets that can be used to decode is dictated by the $(z+1)^{\text{st}}$ fastest worker, i.e., the master can only use the results of secure information packets computed at a given round if at least $z + 1$ workers have completed that round. If for example the z fastest workers have completed round 100 and the $(z+1)^{\text{st}}$ fastest worker has completed round 20, the master can only use the packets belonging to the first 20 rounds. The reason is that the master needs all the keys corresponding to a given round in order to use the secure information packet for decoding. In [7, Lemma 2] we prove that this scheme is optimal, i.e., in private coded computing the master cannot use the packets computed at rounds finished by less than $z + 1$ workers irrespective of the coding scheme.



(a) Scenario 1 with the fastest 13 workers as eavesdropper for GC3P 1 and the slowest workers as eavesdropper for GC3P 2.



(b) Scenario 2 with 13 workers picked at random to be eavesdroppers.



(c) Scenario 3 with 13 workers picked at random to be eavesdroppers.

Fig. 1: Comparison between PRAC and GC3P in different scenarios with $n = 50$ workers and $z = 13$ colluding eavesdroppers for different values of the number of rows m. For each value of m we run 100 experiments and average the results. When the eavesdropper are chosen to be the fastest workers, PRAC has very similar performance to GC3P. When the eavesdroppers are picked randomly, the performance of PRAC becomes closer to this of GC3P when the non adversarial workers are more heterogeneous.

Now that we characterized the task completion delay of PRAC, we can compare it with the state-of-the-art. Secure coded computing schemes that exist in the literature usually use static task allocation, where tasks are assigned to workers a priori. The most recent work in the area is Staircase codes [10], which is shown to outperform all existing schemes that use threshold secret sharing [11]. Therefore, we restrict our focus on comparing PRAC to Staircase codes.

Staircase codes assigns a task of size $b/(k-z)$ row blocks to each worker.[8] Let T_i be the time spent at worker i to compute the whole assigned task. Denote by $T_{(i)}$ the i^{th} order statistic of the T_i's and by $T_{\text{SC}}(n,k,z)$ the task completion time, i.e., time the master waits until it can decode $A\mathbf{x}$, when using Staircase codes. In order to decode $A\mathbf{x}$ the master needs to receive a fraction equal to $(k-z)/(d-z)$ of the task assigned to each worker from any d workers where $k \leq d \leq n$. The task completion time of the master can then be expressed as

$$T_{\text{SC}}(n,k,z) = \min_{d\in\{k,\dots,n\}} \left\{ \frac{k-z}{d-z} T_{(d)} \right\}. \tag{4}$$

Theorem 1. *The gap between the completion time of PRAC and coded computation using staircase codes is lower bounded by:*

$$\mathbb{E}[T_{SC}] - \mathbb{E}\left[T_{PRAC}\right] \geq \frac{bx - \epsilon y}{y(x+y)}, \tag{5}$$

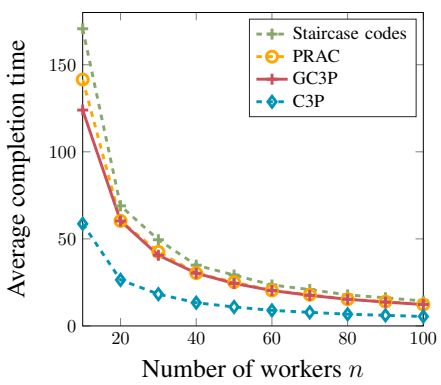
where $x = \dfrac{n-d^*}{E[\beta_{t,w_n}]}$, $y = \dfrac{d^*-z}{E[\beta_{t,h_{d^*}}]}$, w_n *is the index of the slowest worker and* d^* *is the value of* d *that minimizes equation* (4).

Proof. The proof of Theorem 1 is provided in [7]. □

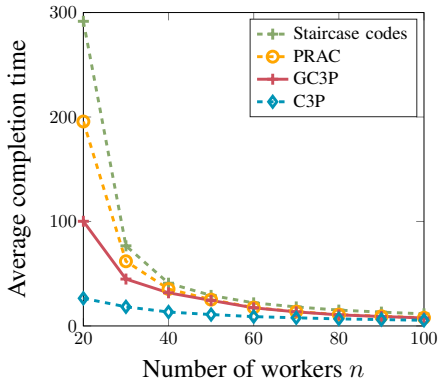
Theorem 1 shows that the lower bound on the gap between secure coded computation using staircase codes and PRAC is in the order of number of row block of A. Hence, the gap between secure coded computation using staircase codes and PRAC is linearly increasing with the number of row blocks of A. Note that, ϵ, the required overhead by fountain coding used in PRAC, becomes negligible by increasing b.

Thus, PRAC outperforms secure coded computation using Staircase codes in heterogeneous systems. The improvement obtained by using PRAC increases when the difference between the computing times of the workers increases. However, Staircase codes can slightly outperform PRAC in the case where the slowest $n-z$ workers are homogeneous, i.e., have compute service times T. Staircase codes outperform PRAC in homogeneous case, because both algorithms are restricted to the slowest $n-z$ workers (see [7]), but PRAC incurs an ϵ overhead of tasks (due to using Fountain codes) which is not needed for Staircase codes.

[8]Note that in addition to n and z, all threshold secret sharing based schemes require a parameter k, $z < k < n$, which is the minimum number of non stragglers that the master has to wait for before decoding $A\mathbf{x}$.



(a) Task completion time as a function of the number of workers with $z = n/4$.



(b) Task completion time as a function of the number of workers with $z = 13$.

Fig. 2: Comparison between PRAC, Staircase codes and GC3P in scenario 1 for different values of the number workers and number of colluding workers. We fix the number of rows to $m = 1000$. For each value of the x-axis we run 100 experiments and average the results. We observe that the difference between the completion time of PRAC and this of GC3P is small for small number of colluding workers and increases with the increase of z.

IV. Performance Evaluation

In this section, we present simulations run on MATLAB, and compare PRAC with the following baselines: (i) Staircase codes [11], (ii) C3P [8] (which is not secure as it is not designed to be secure), and (iii) Genie C3P (GC3P) that extends C3P by assuming a knowledge of the identity of the eavesdroppers and ignoring them. We note that GC3P serves as a lower bound on private coded computing schemes.

In our simulations, we model the computation time of each worker w_i by an independent shifted exponential random variable with rate λ_i and shift c_i, i.e., $F(T_i = t) = 1 - \exp(-\lambda_i(t - c_i))$. We take $c_i = 1/\lambda_i$ and consider three different scenarios for choosing the values of λ_i's for the workers as follows:

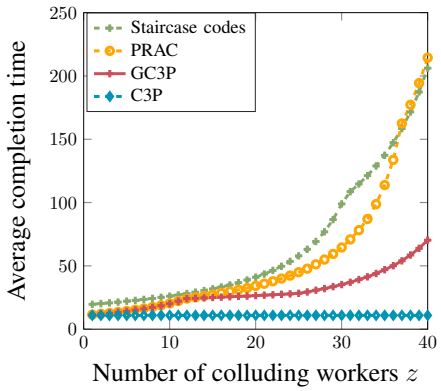
- *Scenario 1*: we assign $\lambda_i = 3$ for half the workers, then we assign $\lambda_i = 1$ for one quarter of the workers and assign $\lambda_i = 9$ for the remaining workers.
- *Scenario 2*: we assign $\lambda_i = 1$ for one third of the workers, the second third have $\lambda_i = 3$ and the remaining workers have $\lambda_i = 9$.
- *Scenario 3*: we draw the λ_i's independently and uniformly at random from the interval $[0.5, 9]$.

When running Staircase codes, we choose the parameter k that minimizes the task completion time for the desired n and z. We do so by simulating Staircase codes for all possible values of $z \leq k \leq n$ and choose the one with the minimum completion time.

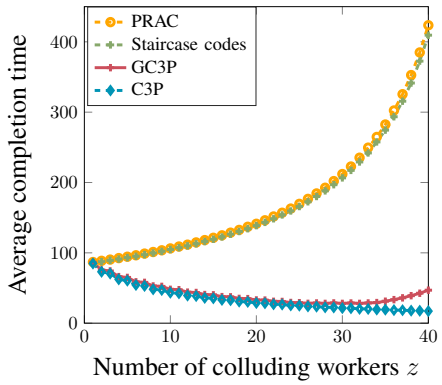
We take $b = m$, i.e., each row block is simply a row of A. The size of each element of A and vector $\mathbf{x}$ are assumed to be 1 Byte (or 8 bits). Therefore, the size of each transmitted packet $p_{t,i}$ is $8 * \ell$ bits. C_i denotes the average channel capacity of each worker w_i and is selected uniformly from the interval $[10, 20]$ Mbps. The rate of sending a packet to worker w_i is sampled from a Poisson distribution with mean C_i.

In Figure 1 we show the effect of the number of rows m on the completion time at the master. We fix the number of workers to 50 and the number of colluding workers to 13 and plot the completion time for PRAC, C3P, GC3P and Staircase codes. Notice that PRAC and Staircase codes have close completion time in scenario 1 and this completion time is far from that of C3P. The reason is that in this scenario we pick exactly 13 workers to be fast ($\lambda_i = 9$) and the others to be significantly slower. Since PRAC assigns keys to the fastest z workers, the completion time is dictated by the slow workers. To compare PRAC to Staircase codes notice that the majority of the remaining workers have $\lambda_i = 1$ therefore pre-allocating equal tasks to the workers is close to adaptively allocating the tasks.

In terms of lower bound on PRAC, observe that when the fastest workers are assumed to be adversarial, GC3P and PRAC have very similar task completion time. However, when the slowest workers are assumed to be adversarial the completion of GC3P is very close to C3P and far from PRAC. This observation is in accordance with the theory (Section III-C). In scenarios 2 and 3 we pick the adversarial workers uniformly at random and observe that the completion



(a) Task completion time as a function of the number of colluding workers for $n = 50$. Computing time of the workers are chosen according to scenario 1.



(b) Task completion time for $n = 50$ workers and variable z. Computing times of the workers are chosen such that the $n - z$ slowest workers are homogeneous.

Fig. 3: Comparison between PRAC and Staircase codes average completion time as a function of number of colluding workers z. Both codes are affected by the increase of number of colluding helpers because their runtime is restricted to the slowest $n - z$ workers. We observe that PRAC outperforms Staircase codes except when the $n - z$ slowest workers are homogeneous.

time of PRAC becomes closer to GC3P when the workers are more heterogeneous. For instance, in scenario 3, GC3P and PRAC have closer performance when the workers' computing times are chosen uniformly at random from $[0.5, 9]$.

In Figure 2, we plot the task completion time as a function of the number of workers n for a fixed number of rows $m = 1000$ and λ_i's assigned according to scenario 1. In Figure 2(a), we change the number of workers from 10 to 100 and keep the ratio $z/n = 1/4$ fixed. We notice that with the increase of n the completion time of PRAC becomes closer to GC3P. In Figure 2(b), we change the number of workers from 20 to 100 and keep $z = 13$ fixed. We notice that with the increase of n, the effect of the eavesdropper is amortized and the completion time of PRAC becomes closer to C3P. In this setting, PRAC always outperforms Staircase codes.

In Figure 3, we plot the task completion time as a function of the number of colluding workers. In Figure 3(a), we choose the computing time at the workers according to scenario 1. We change z from 1 to 40 and observe that the completion time of PRAC deviates from that of GC3P with the increase of z. More importantly, we observe two inflection points of the average completion time of PRAC at $z = 13$ and $z = 37$. Those inflection points are due to the fact that we have 12 fast workers ($\lambda = 9$) and 25 workers with medium speed ($\lambda = 3$) in the system. For $z > 36$, the completion time of Staircase codes becomes less than that of PRAC because the 14 slowest workers are homogeneous. Therefore, pre-allocating the tasks is better than using Fountain codes and paying for the overhead of computations. To confirm that Staircase codes always outperforms PRAC when the slowest $n - z$ workers are homogeneous, we run a simulation in which we divide the workers into three clusters. The first cluster consists of $\lfloor z/2 \rfloor$ fast workers ($\lambda = 9$), the second consists of $\lfloor z/2 \rfloor + 1$ workers that are regular ($\lambda = 3$) and the remaining $n - z$ workers are slow ($\lambda = 1$). In Figure 3(b) we fix n to 50 and change z from 1 to 40. We observe that Staircase codes always outperform PRAC in this setting. In contrast to non secure C3P, Staircase codes and PRAC are always restricted to the slowest $n - z$ workers and cannot leverage the increase of the number of fast workers. For GC3P, we assume that the slowest workers are eavesdroppers. We note that as expected from Section III-C, when the fastest workers are assumed to be eavesdroppers the performance of GC3P and PRAC becomes very close, see also [7, Lemma 2].

V. Related work

Mobile cloud computing is a rapidly growing field with the aim of providing better experience of quality and extensive computing resources to mobile devices. The main solution to mobile computing is to offload tasks to the cloud or to neighboring devices by exploiting connectivity of the devices. With task offloading comes several challenges such as heterogeneity of the devices, time varying communication channels and energy efficiency. We refer interested reader to [8] and references within for a detailed literature on edge computing and mobile cloud computing.

The problem of stragglers in distributed systems is initially studied by the distributed computing community, see e.g., [12], [13] Research interest in using coding theoretical techniques for straggler mitigation in distributed content

download and distributed computing is rapidly growing. The early body of work focused on content download, see e.g., [14], [15] Using codes for straggler mitigation in distributed computing started in [16] where the authors proposed the use of MDS codes for distributed linear machine learning algorithms in homogeneous workers setting.

Following the work of [16], coding schemes for straggler mitigation in distributed matrix-matrix multiplication, coded computing and machine learning algorithms are introduced and the fundamental limits between the computation load and the communication cost are studied, see e.g., [17], [18] and references within for matrix-matrix multiplication, see [19], [20] for machine learning algorithms and [21], [22] and references within for other topics.

Codes for privacy and straggler mitigation in distributed computing are first introduced in [11] where the authors consider a homogeneous setting and focus on matrix-vector multiplication. Privacy and straggler mitigation in non linear distributed computation is considered in [23], [24]. Works on privacy-preserving machine learning algorithms are also related to our work. However, the privacy constraint in this line of work is computational privacy and the proposed solutions do not take stragglers into account, see e.g., [25], [26]

VI. Conclusion

The focus of this paper is to develop a secure edge computing mechanism to mitigate the computational bottleneck of IoT devices by allowing these devices to help each other in their computations, with possible help from the cloud if available. Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation. Focusing on eavesdropping attacks, we designed a private and rateless adaptive coded computation (PRAC) mechanism considering (i) the privacy requirements of IoT applications and devices, and (ii) the heterogeneous and time-varying resources of edge devices. Our proposed PRAC model can provide adequate security and latency guarantees to support real-time mission critical computation at the tactical edge We showed through analysis and MATLAB simulations that PRAC outperforms known secure coded computing methods when resources are heterogeneous.

VII. Acknowledgement

This work was supported in parts by the Army Research Lab (ARL) under Grant W911NF-1820181, National Science Foundation (NSF) under Grants CNS-1801708 and CNS-1801630, and the National Institute of Standards and Technology (NIST) under Grant 70NANB17H188.

References

[1] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.
[2] M. Luby, "Lt codes," in *null*, p. 271, IEEE, 2002.
[3] A. Shokrollahi, "Raptor codes," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2551–2567, 2006.
[4] D. J. MacKay, "Fountain codes," *IEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.
[5] S. Lin and D. Costello, *Error-Correcting Codes*. Prentice-Hall, Inc, 1983.
[6] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.
[7] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. E. Rouayheb, and H. Seferoglu, "PRAC: Private and Rateless Adaptive Coded computation at the edge," *http://eden.rutgers.edu/ rb1033/PRAC.pdf*, 2019.
[8] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, Sept 2018.
[9] J. Lacan and J. Fimes, "Systematic mds erasure codes based on vandermonde matrices," *IEEE Communications Letters*, vol. 8, no. 9, pp. 570–572, 2004.
[10] R. Bitar and S. E. Rouayheb, "Staircase codes for secret sharing with optimal communication and read overheads," in *IEEE International Symposium on Information Theory (ISIT)*, pp. 1396–1400, July 2016.
[11] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2900–2904, IEEE, 2017.
[12] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[13] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments.," in *Osdi*, vol. 8, p. 7, 2008.
[14] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *IEEE International Symposium on Information Theory (ISIT)*, 2012.
[15] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *50th Annual Allerton Conference on Communication, Control, and Computing*, 2012.
[16] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *arXiv preprint arXiv:1512.02673*, 2015.
[17] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, pp. 4403–4413, 2017.
[18] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *arXiv preprint, arXiv:1801.07487*, 2018.

[19] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, pp. 5434–5442, 2017.
[20] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with ldpc codes," *arXiv preprint arXiv:1805.08327*, 2018.
[21] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in 29*th Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 2092–2100, 2016.
[22] Y. Yang, P. Grover, and S. Kar, "Computing linear transformations with unreliable components," *IEEE Transactions on Information Theory*, 2017.
[23] H. Yang and J. Lee, "Secure distributed computing with straggling servers using polynomial codes," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 141–150, Jan 2019.
[24] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv preprint, arXiv:1806.00939*, 2018.
[25] H. Takabi, E. Hesamifard, and M. Ghasemi, "Privacy preserving multi-party machine learning with homomorphic encryption," in 29*th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016.
[26] R. Hall, S. E. Fienberg, and Y. Nardi, "Secure multiple linear regression based on homomorphic encryption," *Journal of Official Statistics*, vol. 27, no. 4, p. 669, 2011.