Differentially Private Selection from Secure Distributed Computing

Ivan Damgård
Aarhus University
Denmark
ivan@cs.au.dk

Hannah Keller
Aarhus University
Denmark
hkeller@cs.au.dk

Boel Nelson
University of Copenhagen
Denmark
bn@di.ku.dk

Claudio Orlandi
Aarhus University
Denmark
orlandi@cs.au.dk

Rasmus Pagh
University of Copenhagen
Denmark
pagh@di.ku.dk

Abstract

Given a collection of vectors $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} \in \{0,1\}^d$, the *selection* problem asks to report the index of an "approximately largest" entry in $\boldsymbol{x} = \sum_{j=1}^{n} \boldsymbol{x}^{(j)}$. Selection abstracts a host of problems—in machine learning it can be used for hyperparameter tuning, feature selection, or to model empirical risk minimization. We study selection under differential privacy, where a released index guarantees privacy for individual vectors. Though selection can be solved with an excellent utility guarantee in the central model of differential privacy, the distributed setting where no single entity is trusted to aggregate the data lacks solutions. Specifically, strong privacy guarantees with high utility are offered in high trust settings, but not in low trust settings. For example, in the popular *shuffle model* of distributed differential privacy, there are strong lower bounds suggesting that the utility of the central model cannot be obtained. In this paper we design a protocol for differentially private selection in a trust setting similar to the shuffle model—with the crucial difference that our protocol tolerates corrupted servers while maintaining privacy. Our protocol uses techniques from secure multi-party computation (MPC) to implement a protocol that: (i) has utility on par with the best mechanisms in the central model, (ii) scales to large, distributed collections of high-dimensional vectors, and (iii) uses $k \geq 3$ servers that collaborate to compute the result, where the differential privacy guarantee holds assuming an honest majority. Since general-purpose MPC techniques are not sufficiently scalable, we propose a novel application of *integer secret sharing*, and evaluate the utility and efficiency of our protocol both theoretically and empirically. Our protocol is the first to demonstrate that large-scale differentially private selection is possible in a distributed setting.

1 Introduction

Differentialy private *selection* of the largest entry in a vector enables data analysis on sensitive datasets—for example announcing the winning candidate in a vote, or identifying a common genetic marker from a set of DNA sequences. While there exist solutions to the selection problem with strong guarantees scaling logarithmically with dimension and independent of the size of vector entries (e.g., [MS20]), they operate in the *central model* of differential privacy, which requires trust in a single party to perform the computation. Existing solutions with weaker trust assumptions, on the other hand, scale poorly or require significantly more noise to maintain privacy.

A pragmatic solution is to aim for a middle ground: distributing trust among multiple parties. This setting is natural when a person trusts a party (e.g., their local hospital) with their data, but not *every* party (e.g., they may not want to share their data with every hospital). In principle, every mechanism in the central model of differential privacy could be simulated in such a distributed setting using techniques for secure multi-party computation (MPC), but that approach is not viable in general because MPC is not yet practical for large-scale general-purpose computations. Steinke [Ste20] introduced a more restricted class of protocols working in the so-called *multi-central model*, in which data holders submit information to k servers, which then communicate and compute the output of the mechanism. An attractive property of this model is that data holders only need to submit a single message to each server, after which no involvement is needed. Nevertheless, techniques such as additive secret sharing allow protocols that have high utility and protect privacy even if $k-1$ servers share their information. However, MPC protocols tolerating $k-1$ corruptions require computationally heavy public-key encryption techniques and are not very efficient. In this work we will therefore work with a slightly weaker notion of privacy: the information gained by any *minority* of the servers is differentially private. This allows the MPC solution to be much more efficient.

A popular approach to differentially private protocols in distributed settings is the *shuffle model* [BEM$^+$17, CSU$^+$19] in which scalable techniques from cryptography are combined with techniques from differential privacy, often allowing utility close to what is possible in the central model. However, existing protocols for selection use private summation, which is known to require much more noise than selection. It is likely that there is a fundamental obstacle to achieving better utility for selection in the shuffle model, due to the lower bound of [CU21] which holds for a wide class of mechanisms in the shuffle model. Another general tool for distributed differential privacy, *secure aggregation* [GX17], faces the same problem, namely that the magnitude of noise needs to grow polynomially with the dimension d of the input vectors. Finally, we mention *local differential privacy* (LDP) [DJW13], in which each input vector is independently made differentially private, and where the magnitude of noise grows polynomially in the number n of input vectors.

Given that existing distributed methods for the selection problem are far from matching what is possible in the central model, and since we know that *in principle* it is possible to simulate the central model with MPC techniques, Steinke [Ste20] suggests to solve selection via an MPC implementation of argmax on secret-shared sums, but states that further investigation about the practicality is needed. In this work we perform such an investigation, modifying the approach in several ways to achieve the best fit with scalable MPC techniques. The contributions of this work are as follows:

- We present the Noise-and-round mechanism (Section 2), a distributed differentially private selection algorithm with utility guarantees close to the best algorithms in the central model.
- We introduce the first demonstration of the multi-central model for the selection problem using MPC techniques (Section 3).
- We design a new combination of integer secret sharing and existing MPC techniques which is tailored to perform a secure and efficient distributed computation of differentially private selection. In particular, this allows non-interactive truncation of input data so that approximate comparisons can be performed more efficiently than previously known.
- We provide an empirical evaluation of the utility and scalability of Noise-and-round using both synthetic and real-world data for the 3-servers case (Section 4).

Problem formulation. The selection problem is perhaps the simplest instance of "heavy hitters," a problem ubiquitous in data analysis and machine learning. Given a collection of vectors $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} \in \{0,1\}^d$ it asks to report the index of an "approximately largest" entry in $\boldsymbol{x} = \sum_{j=1}^{n} \boldsymbol{x}^{(j)}$. More precisely, the task is to report an index i such that $x_i \geq \max_\ell(x_\ell) - \alpha n$, where $\alpha \in (0,1)$ is an approximation parameter specifying the (additive) error within which x_i is largest. This problem is a special case of general heavy hitters problems, which asks for the most frequently occurring elements in a multiset.

Differential privacy. Differential privacy [DMNS06] formalizes the worst-case information leakage of any output from an algorithm. Given two neighboring datasets as input differential privacy limits how much the output distributions can differ. We say that a pair of datasets are neighboring, denoted $\boldsymbol{x} \sim \boldsymbol{x}'$, if and only if $\boldsymbol{x}$ and $\boldsymbol{x}'$ differ on exactly one element. In this paper, we work in the bounded setting where the dataset's size is fixed.

Algorithm 1 Noise-and-round

```
1: Input: $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} \in \{0,1\}^d$
2: Parameters: $\varepsilon > 0$, $\gamma \geq 1$, $\Delta \geq 0$
3: sample $\boldsymbol{\eta} \sim \text{Geometric}(1 - e^{-\varepsilon/2})^d$
4: $\boldsymbol{w} \leftarrow \text{round}_\Delta((\sum_{j=1}^d \boldsymbol{x}^{(j)} + \boldsymbol{\eta})/\gamma)$
5: return $\arg\max_i(\boldsymbol{w}_i)$
```

Definition 1.1 ([DMNS06] (ε, δ)-differential privacy)**.** A randomized mechanism $\mathcal{M}$ satisfies $(\varepsilon, \delta)-$ differential privacy if and only if for all pairs of neighboring datasets $\boldsymbol{x} \sim \boldsymbol{x}'$ and all set of outputs Z we have $\Pr[\mathcal{M}(\boldsymbol{x}) \in Z] \leq e^\varepsilon \Pr[\mathcal{M}(\boldsymbol{x}') \in Z] + \delta$. If $\mathcal{M}$ satisfies (ε, 0)-DP we say that it satisfies ε-differential privacy.

Technical overview. We first describe our approach in the central model and then extend to the distributed setting. The technique is rather standard, but with a couple of deviations: first, while [Ste20, DR14] suggests to use Laplace noise, we show that it suffices to use one-sided, geometric noise when computing the noisy argmax. Second, we show that (not surprisingly) the protocol is robust to scaling and rounding before taking argmax, which helps the efficiency of the MPC protocol.

The bottleneck in the secure computation protocol is the comparisons required to compute argmax. For this we use state-of-the-art protocols from [EGK$^+$20]. These must be supplied initially with correlated randomness and are constructed as protocols for dishonest majority. However, we assume k servers with t semi-honest corruptions where $t < k/2$. Therefore, with the help of all servers, we can preprocess the correlated randomness using the honest majority protocol from [ACD$^+$19], after which the first $t+1$ servers run the protocol from [EGK$^+$20]. Finally, we let data owners supply inputs as secret shares over the integers. This allows the servers to truncate the input without interaction while introducing only a small error; then the comparisons can work over fewer bits and hence be more efficient.

2 Algorithm in the central model

In this section we analyze Algorithm 1, which solves selection in the central model and is well-suited for being extended to an efficient secure multi-party computation protocol (described in Section 3). The algorithm is a variant of the well-known "report noisy argmax" approach to selection, which has been proposed as a candidate algorithm on which to base an MPC implementation [Ste20].

Compared to a plain noisy argmax approach we make two modifications that will improve efficiency of the MPC protocol: 1) Use one-sided, geometric error, and 2) allow the argmax to be based on rounded values. Rounding is controlled by a parameter Δ, such that for a rational number w, $\text{round}_\Delta(w)$ denotes an integer value (possibly the output of a randomized algorithm) that differs from w by at most Δ, and for inputs $\frac{x+\eta}{\gamma}$ and $\frac{\bar{x}+\eta}{\gamma}$ with $|x - \bar{x}| \leq 1$, using the same internal randomness for both inputs, satistifies:

$$\left|\text{round}_\Delta\left(\frac{x+\eta}{\gamma}\right) - \text{round}_\Delta\left(\frac{\bar{x}+\eta}{\gamma}\right)\right| \leq 1 \quad . \tag{1}$$

When applied to a vector $\boldsymbol{x}$, $\text{round}_\Delta(\boldsymbol{x})$ is computed by rounding independently on each coordinate. Looking ahead to the distributed implementation of the algorithm, allowing this rounding error will allow us to perform truncation using a simple and efficient method. Proof in supplementary material.

Lemma 1. *Algorithm 1 is ε-differentially private.*

Lemma 2. *Algorithm 1 has error at most $2\gamma\Delta + 4\ln(d)/\varepsilon$ with probability at least $1 - 1/d$.*

Proof. By a union bound, $\Pr[\|\boldsymbol{\eta}\|_\infty > 4\ln(d)/\varepsilon] \leq d \Pr[\boldsymbol{\eta}_i > 4\ln(d)/\varepsilon] < 1/d$. Let $\mathcal{M}(\boldsymbol{x})$ denote the output of Algorithm 1, where $\boldsymbol{x} = \sum_{j=1}^d \boldsymbol{x}^{(j)}$ is the sum of the input vectors. We want to argue that the error $|\boldsymbol{x}_{\mathcal{M}(\boldsymbol{x})} - \max_\ell(\boldsymbol{x}_\ell)|$ is not too large. Abbreviating $i = \mathcal{M}(\boldsymbol{x})$, $j = \arg\max_\ell(\boldsymbol{x}_\ell)$,

Algorithm 2 Relaxed-noise-and-round (The "Ideal Functionality")

```
1: Input: $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)} \in \{0,1\}^d$
2: Parameters: $p$ (noise parameter), $c$ (bits to truncate), $k$ (number of servers), $t$ (upper bound on
   corrupted servers),
3: for all $j \in [k]$ sample $\boldsymbol{r}^{(j)} \sim \mathrm{NB}^d(1/(k-t), p)$
4: $\boldsymbol{z} \leftarrow \sum_{i \in [n]} \boldsymbol{x}^{(i)} + \sum_{j \in [k]} \boldsymbol{r}^{(j)}$
5: $\boldsymbol{w} \leftarrow \mathrm{round}_\Delta(\boldsymbol{z}/2^c)$
6: Output: $\arg\max_i(w_i)$
7: Leakage: $\boldsymbol{r}^{(j)}$ for $j \in [t]$ (capturing that the corrupted parties contribution to the noise are
   known to the adversary.)
```

and using that entries in $\boldsymbol{\eta}$ are non-negative, we have

$$\begin{aligned}
\mathrm{round}_\Delta\left(\frac{\boldsymbol{x}_j + \boldsymbol{\eta}_j}{\gamma}\right) \le \mathrm{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) &\Rightarrow \frac{\boldsymbol{x}_j + \boldsymbol{\eta}_j}{\gamma} - \Delta \le \frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma} + \Delta \\
&\Rightarrow \boldsymbol{x}_j + \boldsymbol{\eta}_j - (\boldsymbol{x}_i + \boldsymbol{\eta}_i) \le 2\gamma\Delta \\
&\Rightarrow |\boldsymbol{x}_{\mathcal{M}(\boldsymbol{x})} - \max_\ell(\boldsymbol{x}_\ell)| \le 2\gamma\Delta + \|\boldsymbol{\eta}\|_\infty \ . \qquad \square
\end{aligned}$$

3 Secure computation of differentially private selection

As it is common in the MPC literature, we first describe *what* we want to achieve in the form of an idealized algorithm, as it if was executed by some trusted third party—usually referred to as the "ideal functionality". This algorithm formally captures the computation that the distributed protocol will perform, as well as what kind of information is leaked to the adversary, while hiding the details on *how* the distributed protocols achieves this result. This ideal functionality, provided in Algorithm 2, has a small deviation from Algorithm 1; in particular, it adds a larger amount of noise sampled from a negative binomial distribution (some of which is leaked). Such distributed addition of noise has been used before in similar settings [GX17]. The increased level of noise allows us to perform a very simple and efficient distributed noise generation. Moreover, the noise leaked by the functionality is used to capture the fact that, in the distributed implementation of the algorithm, up to t servers might be corrupted by a semi-honest adversary. We use $[n]$ to denote the set $\{1, \dots, n\}$.

Lemma 3. *Algorithm 2 with $p = 1 - e^{-\varepsilon/2}$ and $\gamma = 2^c$ is ε-differentially private, even if the leakage is considered part of the output. It has error at most $2\gamma\Delta + 16\ln(d)/\varepsilon$ with probability at least $1 - 2/d$.*

Proof. By symmetry we can assume that the leakage consists of the noise added by the first t parties, i.e., $\boldsymbol{r}^{(j)}$ for $j \in [t]$. Consider any fixed value of the leaked noise vectors—we will argue that the algorithm is ε-differentially private under the distribution induced by the remaining $k - t$ noise vectors. As before, let $\boldsymbol{x} = \sum_{j=1}^d \boldsymbol{x}^{(j)}$. After Line 4 we have

$$\boldsymbol{z} = \boldsymbol{x} + \sum_{j \in [k]} \boldsymbol{r}^{(j)} = \left(\boldsymbol{x} + \sum_{j \in [t]} \boldsymbol{r}^{(j)}\right) + \sum_{j \in [k] \setminus [t]} \boldsymbol{r}^{(j)},$$

where $\boldsymbol{\eta} = \sum_{j \in [k] \setminus [t]} \boldsymbol{r}^{(j)} \sim \mathrm{Geometric}(p)^d$ since it is a sum of $k - t$ negative binomials $\mathrm{NB}(\frac{1}{k-t}, p)$ (see e.g. [GX17]). Since $p = 1 - e^{-\varepsilon/2}$ this means that Algorithm 2 has the same output distribution as Algorithm 1 applied to an input with sum $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \tilde{\boldsymbol{\eta}}$, where $\tilde{\boldsymbol{\eta}} = \sum_{j \in [t]} \boldsymbol{r}^{(j)}$ is the additional noise added by the first t parties. Since neighboring input sums $\boldsymbol{x} \sim \boldsymbol{x}'$ translate to neighboring input sums $\tilde{\boldsymbol{x}} \sim \tilde{\boldsymbol{x}}'$ we conclude that Algorithm 2 is ε-differentially private.

Algorithm 3 Primitives for Integer Secret Sharing

```
1: Addition. $[z]_{\mathbb{Z}} \leftarrow [x]_{\mathbb{Z}} + [y]_{\mathbb{Z}}$ means that each server $S_i$ locally adds their shares, i.e., $z_i = x_i + y_i$
   leading to $z = x + y$.
2: Truncation. $[y]_{\mathbb{Z}} \leftarrow \mathtt{trunc}_\Delta([x]_{\mathbb{Z}}, c)$ means that each server $S_i$ locally computes $y_i = \lfloor x_i/2^c \rceil$
   for all $i \in [h]$, removing the least significant $c$ bits from each share $x_i$ and rounding, leading to
   $x/2^c - \Delta \leq y \leq x/2^c + \Delta$, for a value $\Delta$ analyzed below.
3: Conversion. $[y]_{2^a} \leftarrow \mathtt{convert}([x]_{\mathbb{Z}})$ means that each server $S_i$ locally computes $y_i = x_i \bmod$
   $2^a$, leading to $y = x$ assuming $x \leq 2^a$ This is correct because $\sum_{i \in [h]}(x_i \bmod 2^a) \bmod 2^a =$
   $\sum_{i \in [h]} x_i \bmod 2^a = x \bmod 2^a$.
```

Abbreviating $i = \mathcal{M}(\tilde{\boldsymbol{x}})$ and $j = \arg\max_\ell(\boldsymbol{x}_\ell)$ we have, similar to the proof of Lemma 2,

$$\begin{aligned}
\text{round}_\Delta\left(\frac{\tilde{\boldsymbol{x}}_j + \boldsymbol{\eta}_j}{\gamma}\right) \leq \text{round}_\Delta\left(\frac{\tilde{\boldsymbol{x}}_i + \boldsymbol{\eta}_i}{\gamma}\right) &\Rightarrow \tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i \leq 2\gamma\Delta + \boldsymbol{\eta}_i - \boldsymbol{\eta}_j \\
&\Rightarrow \boldsymbol{x}_j - \boldsymbol{x}_i \leq 2\gamma\Delta + \boldsymbol{\eta}_i - \boldsymbol{\eta}_j - \tilde{\boldsymbol{\eta}}_j + \tilde{\boldsymbol{\eta}}'_i \\
&\Rightarrow |\boldsymbol{x}_{\mathcal{M}(\boldsymbol{x})} - \max_\ell(\boldsymbol{x}_\ell)| \leq 2\gamma\Delta + 2\left\|\boldsymbol{\eta}\right\|_\infty + 2\left\|\tilde{\boldsymbol{\eta}}\right\|_\infty \ .
\end{aligned}$$

Since $\|\boldsymbol{\eta}\|_\infty > 4\ln(d)/\varepsilon$ and $\|\tilde{\boldsymbol{\eta}}\|_\infty > 4\ln(d)/\varepsilon$ each happen with probability at most $1/d$ (the latter because the sum is dominated by a geometric distribution with parameter p) we are done. $\square$

3.1 Secret-sharing: notation and techniques

Our distributed protocol is performed by k-servers denoted by $\mathcal{S} = \{S_1, \ldots, S_k\}$. We assume that at most t of them are corrupted by a semi-honest adversary (i.e., they follow the protocol specifications but then will try to infer more information by collecting their data) with $k = 2 \cdot t + 1$. We let $h = k - t = t + 1$ be the minimum number of guaranteed honest servers. As it is common in the secure multipary computation literature, we assume a single, monolithic adversary that controls all corrupted parties and collects all their internal states. This can be thought of as an adversary who has installed "spyware" on the corrupted servers: the adversary is able to observe everything that the servers observe, but not to change the code they are running. Finally, the servers will have slightly asymmetric roles in the protocol. The first h servers are called the *computation servers*, whereas the last t servers are called the *supporting servers* (note that by our assumptions on k and t, at least one computation server is guaranteed to be honest, while we can tolerate that all the supporting servers might be dishonest).

We use an additive integer secret sharing scheme among the computing servers $S_1, S_2, \ldots, S_h$. We use $[x]_{\mathbb{Z}}$ to denote a secret sharing of some integer x, consisting of shares $x_1, \ldots, x_h \in \mathbb{Z}$ such that $\sum_{i=1}^h x_i = x$. For every $i \in [h]$, S_i has x_i. In order to securely share an ℓ-bit long secret, we need that the shares are chosen uniformly at random among integers with $\ell + \kappa$ bits. This results in statistical security with negligible security error $2^{-\kappa}$ against any adversary, even if computationally unbounded. That is, the security of our distributed protocol does not rely on any computational assumption. Our distributed protocol performs additions and truncation of integer secret sharings, which are detailed in Algorithm 3.

Truncation error. Here we analyze $\Delta = |x/2^c - \sum_{i \in [h]} \lfloor x_i/2^c \rceil|$, the possible error incurred by truncation. The error depends on h, the number of shares of the secret. Consider the case of $h = 2$: if the input is secret shared among two servers, at most one carry bit may be missed when truncating the lower order bits. To generalize to larger h, first observe that division and rounding incurs an error of at most $e_i = |x_i/2^c - \lfloor x_i/2^c \rceil| \leq 1/2$. For shared integer x and shares $x_1, \ldots, x_h$, when we divide $x/2^c$, we can write the result as $x_1/2^c + x_2/2^c + \cdots + x_h/2^c$. Then we can formulate the total error $\Delta = |\sum_{i \in [h]} x_i/2^c - \lfloor x_i/2^c \rceil| \leq \sum_{i \in [h]} e_i \leq h/2$ by the triangle inequality and then applying our bound for e_i. Notice that $\mathtt{trunc}_\Delta([x]_{\mathbb{Z}}, c)$ exactly implements $\text{round}_\Delta(x/2^c)$ with $\Delta = h/2$.

3.2 A secure and differentially private distributed protocol for selection

We are finally ready to describe, in Algorithm 4, a secure distributed implementation of the differentially private mechanism from Algorithm 2 (the "ideal functionality"). The protocol proceeds

Algorithm 4 Distributed-noise-and-round (The MPC Protocol)

```
1: Input: Integer secret-sharings $[\boldsymbol{x}^{(1)}]_{\mathbb{Z}}, \ldots, [\boldsymbol{x}^{(n)}]_{\mathbb{Z}}$ representing values in $\{0,1\}^d$
2: Parameters: $p$ (noise parameter), $c$ (bits to truncate), $k$ (number of servers), $t$ (upper bound on
   corrupted servers), $\kappa$ (security parameter used in integer secret sharing), $a = \log(n) - c + 1$ (bits
   for modular secret sharing)
3: $[\mathrm{corr}]_{2^a} \leftarrow \texttt{preprocessing}(S_1, \ldots, S_k)$
4: $\forall j \in [k]$, $S_j$ samples $\boldsymbol{r}^{(j)} \sim \mathrm{NB}^d(1/(k-t), p)$
5: $\forall j \in [t+2, k]$, $S_j$ secret-shares $\boldsymbol{r}^{(j)}$ as $[\boldsymbol{r}^{(j)}]_{\mathbb{Z}}$ and send the corresponding shares to $S_1, \ldots, S_h$.
6: $S_1, \ldots, S_h$ evaluate $[\boldsymbol{z}]_{\mathbb{Z}} = \left[\sum_{i \in n} \boldsymbol{x}^{(i)} + \sum_{j \in [k]} \boldsymbol{r}^{(j)}\right]_{\mathbb{Z}}$.
7: $S_1, \ldots, S_h$ compute $[\boldsymbol{y}]_{\mathbb{Z}} = \texttt{trunc}_{\Delta}([\boldsymbol{z}]_{\mathbb{Z}}, c)$
8: $S_1, \ldots, S_h$ convert $[\boldsymbol{y}]_{2^a} \leftarrow \texttt{convert}([\boldsymbol{y}]_{\mathbb{Z}})$
9: $S_1, \ldots, S_h$ execute $[o]_{2^a} \leftarrow \texttt{ArgMax}([\boldsymbol{y}]_{2^a}, [\mathrm{corr}]_{2^a})$.
10: Output: Open and output $o = \mathrm{argmax}_{j \in [d]} [\boldsymbol{y}]_{2^a}$
```

as follows: In Line 4, all servers (computing and supporting) locally sample noise according to the negative binomial distribution, with parameter inversely proportional to the number of honest parties. The supporting servers need now to share their noise contribution to the computing servers in Line 5 (this can be done assuming using shares of size $\kappa + \log(n)$ assuming $\log(n)$ as an upper bound on the noise magnitude). In Line 6 the computing servers exploit the linear nature of the secret sharing scheme to locally aggregate the input vectors and all noise contributions, in secret shared form. To do so, they each add all input shares and noise shares received from the supporting parties, as well as their own randomly generated noise. To increase efficiency the result is then truncated in Line 7, by removing the lowest c bits (essentially dividing every value by 2^c). The secret-sharing are then converted from integer to modular form in Line 8, to be compatible with the the secure `ArgMax` protocol which is invoked in Line 9. This protocol consumes correlated randomness which is generated by all servers during a preprocessing phase in Line 4. More details on how the `ArgMax` protocol and its preprocessing are implemented are given in Section 3.3.

Correctness. We argue that the output of Algorithm 4 has the same distribution as the one in the ideal functionality specified in Algorithm 2. First, note that the inputs are a secret shared version of the same inputs for the ideal functionality. In Line 4, noise is drawn according to the same distribution specified in Line 3 of Algorithm 2. Secret sharing and addition performed in Lines 5 and 6 correctly add the input values and random samples. In Line 7 we truncate using the secret shared version of `trunc` with the same output in secret shared form, and in Line 8 the conversion to secret sharing over a ring from Algorithm 3 is applied, and a is chosen to be of appropriate size for this conversion to be lossless. Lastly, correctness of the `ArgMax` protocol used in Line 9 guarantees that the algorithm outputs the correct argmax value.

Security. Intuitively, security of the distributed protocol follows from the fact that the entire computation is performed over secret-shared values and that all employed sub-protocols are secure. More precisely, as it is common in the MPC literature, we can prove that the protocol is secure by providing a simulator that, given access to the input/output of the ideal functionality (including the leakage) simulates the view of the corrupted servers in the execution of the protocol. In our case the simulator, which takes as input the set of corrupted servers, and their inputs/outputs, will simulate the view of the corrupted servers essentially by running an execution of the real protocol but where the shares of all the honest parties are set to some dummy value (e.g., 0). The view of the corrupted servers contains all their shares and all the messages that they receive from the honest servers. This includes the messages that they receive from the honest servers in the `preprocessing` phase which, by assumption on the security of the `preprocessing` protocol, can be efficiently simulated. The view contains also the shares of the noise generated by honest supporting servers in Line 4 which can be simulated (with statistical security $2^{-\kappa}$) by picking uniform random shares of the same size $\log(n) + \kappa$ bits as in the protocol. The Lines 6-8 only consist of local computation and can therefore be trivially simulated. Note however that, due to the local addition of the noise by the computing servers, the shares of $[\boldsymbol{y}]_{2^a}$ at the end of Line 8 might not be uniformly random. This does not matter, since the shares are never revelaed but instead used as input in the secure `ArgMax` sub-protocol, which secure as shown in [EGK$^+$20] (and in particular, internally, only reveals results of secure comparison protocols). Overall, the protocol in Algorithm 4 can be efficiently simulated with statistical error $2^{-\kappa}$ (due to the statistical security of the integer secret-sharing scheme) having access to the ideal functionality specified in Algorithm 2. This leads to the following:

Corollary 3.1. *Algorithm 4 with $p = 1 - e^{-\varepsilon/2}$ is $(\varepsilon, 2^{-\kappa})$-differentially private in the view of an adversary that semi-honestly corrupts any t servers. It has the same error as Algorithm 2.*

3.3 Details on the `ArgMax` protocol and preprocessing

There are multiple possible approaches for computing the exact argmax within an MPC protocol. We choose the state-of-the-art solution, which is to use a tree data structure, where the maximum of two values is compared to the maximum of two other values in each step. This approach requires $\mathcal{O}(d)$ comparisons when finding the argmax of d values. To perform the comparisons we use in turn the integer comparison protocol of [EGK$^+$20], which requires the randomness $[\mathrm{corr}]_{2^a}$ generated in the precomputation phase.

We proceed now to describe the necessary correlated randomness to execute the comparison from [EGK$^+$20], and how to generate it: we let all k servers collaborate in producing the correlated randomness. This allows us to achieve unconditional security (thanks to the honest majority assumption) but also to achieve high efficiency using the the protocol from [ACD$^+$19]. This protocol allows us to perform MPC over $\mathbb{Z}_{2^a}$. In a nutshell, their idea is to consider a so called Galois extension R of $\mathbb{Z}_{2^a}$. In the ring R we can do Shamir-style secret sharing (of values in $\mathbb{Z}_{2^a}$) and follow the standard blueprint for honest majority MPC, to perform secure addition and multiplication. This implies an overhead factor $\log_2(k)$, which is necessary as Shamir-style secret sharing cannot be done over $\mathbb{Z}_{2^a}$ directly.

The correlated randomness needed by the protocol from [EGK$^+$20] consists of shared random numbers modulo 2^a, together with the bits in these numbers, also in shared form. Clearly, if we can create shared random bits $[b_0]_{2^a}, ..., [b_{2^{a-1}}]_{2^a}$, this would be sufficient. Namely, if we let r be the number with binary expansion $b_0, b_1, ..., b_{2^{a-1}}$, then using only local computation we can construct

$$[r]_{2^a} = \sum_{i=0}^{a-1} 2^i \cdot [b_i]_{2^a} \quad .$$

In order to get a random shared bit, we can use a trick suggested in [DEF$^+$19]. It was shown there how to generate a random shared bit using secure arithmetic modulo a 2-power, at the cost of a constant number of secure multiplications. This will generate a sharing $[c]_R$, where c is the random bit and $[\cdot]_R$ refers to the secret-sharing scheme from [ACD$^+$19].

Finally, $[c]_R$ can be converted to $[c]_{2^a}$ using only local computation. Namely, if we let $\lambda_1, ..., \lambda_h$ be the Lagrange coefficients one would use to reconstruct a secret over R, and $s_1, ..., s_h$ be the shares of c held by the first h servers, we would have $c = \sum_{i=1}^{h} \lambda_i s_i$. So we can think of the $\lambda_i s_i$-values as additive shares of c. Each such share is an element from R, but it can be represented as a vector of $\log_2(k)$ numbers from $\mathbb{Z}_{2^a}$. Since addition in R is component-wise addition, it turns out that each server can keep only one number from its additive share, discard the rest, and the result will be $[c]_{2^a}$.

To conclude, note that all three protocols in [DEF$^+$19], [EGK$^+$20] and [ACD$^+$19] were originally presented for the malicious security setting, but since we deal with semi-honest corruptions their protocol can be greatly simplified in our setting.

The 3 servers case. We note that our protocol can be highly simplified in the case of $k = 3$. Under the assumption of honest majority this gives $h = 2$ and $t = 1$. Thus we have 2 computing servers and a single supporting server. This means that in Line 4 of the protocol we can simply have the supporting server act as a "dealer" and produce the correlated randomness locally, and then secret share it among the computation servers, instead of having to run a secure protocol among all 3 servers to generate the correlated randomness. This still guarantees security since if the dealer is corrupted then both of the computing servers must be honest (by assumption on $t \leq 1$).

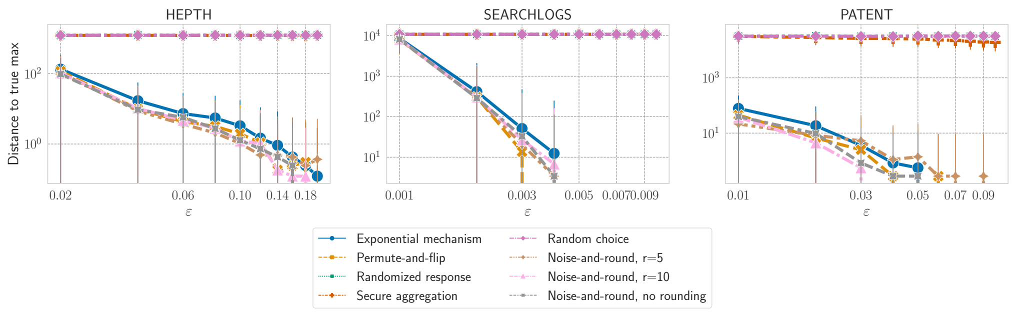


Figure 1: Absolute difference between the real chosen argmax on a log-log scale. Lower is better.

4 Empirical evaluation

Inspired by the evaluation of the state-of-the-art differentially private selection algorithm Permute-and-flip [MS20], we run our benchmarks on the real-world data from DPBench [HMM+16]. Specifically, we use the same five representative datasets (Table 1, full table in Appendix A.4), and discretize them to $d = 1024$ as in [MS20] . To show the scalability of our MPC protocol, we also benchmark performance using synthetic data.

Utility. We implement and run our utility benchmarks using Python 3.11.3, measuring error for 1000 runs as the absolute difference between the true argmax value, and the one chosen by the algorithm. As there are no direct comparisons of differentially private algorithms that use the same trust model (the multi-central model), we compare to differentially private algorithms from both the central model (with stronger trust assumptions), and the local model (with weaker trust assumptions). Representing the best known error for the centralized model, we show Permute-and-flip, as well as the Exponential mechanism [MT07]. For the local model, we compare to bitwise Randomized respone [War65], as used in RAPPOR [EPK14]. As a worst case comparison we also show the error of uniformly at random reporting an index as argmax. Lastly, we show the error of using MPC to compute argmax, without guaranteeing differential privacy, via the use of Secure aggregation [GX17].

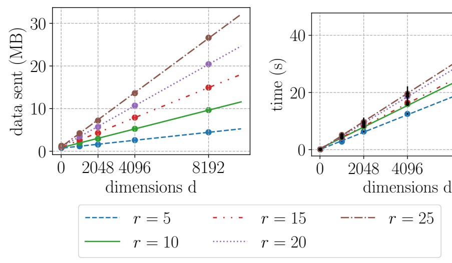
In Figure 1, we highlight the error by varying ε and r on three of the datasets from DPBench, for all datasets see Appendix A.4. We expect Noise-and-round to perform worse than the centralized algorithms (Permute-and-flip, Exponential mechanism) due to their advantage following from their trust model, and better than the local algorithm (Randomized respone) and a purely random choice. As we can see, Noise-and-round performs similar to Permute-and-flip, and better than the Exponential mechanism. When ε increases, error decreases and subsequently reaches 0 (note that the line disappears because of the log-scale). Interestingly, low values for ε cause Randomized respone and Secure aggregation aggregation to perform similar to the completely random choice.

Runtime and communication. The bottleneck for MPC in both time and communication lies in the computation of argmax using comparison operations, so we benchmark this part of the protocol. All benchmarks were carried out on AWS t3.xlarge instances, using MP-SPDZ [Kel20] to implement the protocol in the 3 servers case. In our experiments we vary the input dimensions (d), and the remaining bits (r). We report our results including preprocessing such as multiplication triples, and all time measurements reported are the average of ten executions for the same computation, with standard deviation included either in written form or as error bars for each plot.

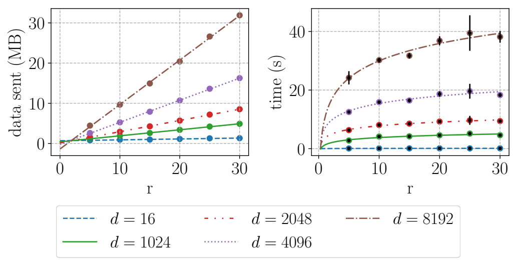
For each of three datasets from DPBench, we report the maximum value in each dataset, the number of bits necessary to represent integers in this range, as well as the runtimes and data sent in Table 1. Notice that while communication scales linearly in the number of bits necessary to represent the data, the time necessary for the evaluations are very close, and the variance in measurements is quite high. The last row in the table reports the necessary time and data necessary when truncating every entry in the dataset to 5 bits using our approach. Note that, due to security of MPC protocols, the runtime of the protocol cannot depend on the actual values that are being computed upon, but only their size. Therefore, the benchmarks after truncation are agnostic of which dataset we start from.

Table 1: Benchmark results for given input

Dataset	Max value (n)	# Bits	Results: Time (s)	Results: Data sent (MB)
PATENT	59602	16	3.83, std=0.18	2.75
SEARCHLOGS	11160	14	3.79, std=0.14	2.48
HEPTH	1571	11	3.80, std=0.11	2.07
Truncated, $\alpha = 0.125$	31	5	2.80, std=0.10	1.78



(a) Communication and time for d in range 16-8192.



(b) Communication and time for r in range 5-30.

Figure 2: MPC overhead, scaling with dimensions and remaining bits; lower is better

The time and communication reported in the last line of the table correspond to the utility reported for $r = 5$ in Figure 1, and the utility of the approach without truncation is reported as well. We observe that by truncating values, the time and communication necessary for these comparisons is significantly reduced. Practitioners may choose how many bits to truncate based on their utility and time requirements, as well as the available computational resources.

The average time required per data point d and power of two in the range r is 0.33 ms, and the average communication is 1.09 kB. Note that while the communication scales linearly in d and r, time scales linearly in d but logarithmically in r. For the chosen range, the complexity can be approximated as linear in r as well.

For synthetic data, the evaluated ring moduli $2^5, 2^{10}, 2^{15}, 2^{20}$, and 2^{25} could correspond either to different value ranges in a dataset before truncation or the resulting range of values after truncation. Based on experiments using synthetic data with sizes 16, 1024, 2048, 4096, and 8192, Figure 2 confirms the linear growth of necessary time and communication in d, as well as the logarithmic growth of time and linear growth in required communication in r. As expected, the savings in cost and communication by performing truncation increases with the size of the dataset and the range of values. Truncating even a few bits results in significant savings in communication and time, particularly when the dataset has several thousands of entries.

5 Related work

The exponential mechanism, as well as "report-noisy-max" [DR14], offer asymptotically optimal solutions to the selection problem in the central model. A mechanism with better constant factors is *permute-and-flip* introduced by [MS20]. We compare with their work by evaluating selection on the same benchmarking datasets and achieve comparable utility using the weaker trust assumptions of multi-central differential privacy.

The setting where data is not, and cannot, be gathered by a central entity, was a motivation for *local differential privacy* [DJW13], where a differentially private function of each participant's data is released. One such protocol for binary data is the classical *randomized response* protocol by Warner [War65]. We can apply randomized response to each bit of a binary vector (splitting the privacy budget), as seen for example in [EPK14], which allows us to estimate the sum of vectors with an error proportional to $\sqrt{n}$, where n is the number of vectors.

Recent work [BEM+17, EFM+19, CSU+19, Ste20] has increasingly focused on models of differential privacy that lie between the central and local models. The shuffle model [BEM+17, CSU+19] is built on trust assumptions that are weaker than the central model, in particular a trusted shuffler, while achieving good utility for some classes of functions. However, [CU21] show an exponential separation between the central and (robust) shuffle models for the selection problem, motivating the need for alternative models.

Compared to the shuffle model, the multi-central model distributes the computation between multiple servers, as opposed to relying on the inputs being sent using an anonymous channel (e.g., using onion routing [DMS04]). The first work to consider the combination of differential privacy and MPC is [DKM+06], which focuses on distributed noise generation; however, their original work focuses on malicious adversaries, while we operate in the semi-honest security model. Some related works focus on replacing the trusted aggregator in DP with an MPC protocol for a variety of computations, while we focus on selection. One particularly prominent application is secure aggregation [GX17, BIK+17, MPBB19, AG21], used for example in federated learning, which lends itself to the use of MPC for differentially private computations and has been implemented in practice. Secure aggregation reveals a noisy sum of inputs and requires larger error than our approach, which reveals only the output.

References

[ACD+19] Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 471–501. Springer, Heidelberg, December 2019.

[AG21] Apple and Google. Exposure Notification Privacy-preserving Analytics (ENPA). White paper, 2021.

[BEM+17] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 441–459, New York, NY, USA, 2017. Association for Computing Machinery.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017.

[CSU+19] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed Differential Privacy via Shuffling. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, Lecture Notes in Computer Science, pages 375–403, Cham, 2019. Springer International Publishing.

[CU21] Albert Cheu and Jonathan Ullman. The limits of pan privacy and shuffle privacy for learning and estimation. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1081–1094, New York, NY, USA, 2021. Association for Computing Machinery.

[DEF+19] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120. IEEE Computer Society Press, 2019.

[DJW13] John C. Duchi, Michael I. Jordan, and Martin J. Wainwright. Local Privacy and Statistical Minimax Rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438, 2013.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 486–503. Springer, Heidelberg, May / June 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, number 3876 in Lecture Notes in Computer Science, pages 265–284. Springer Berlin Heidelberg, 2006.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. Technical report, Defense Technical Information Center (DTIC), 2004.

[DR14] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[EFM$^+$19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 2468–2479. Society for Industrial and Applied Mathematics, 2019.

[EGK$^+$20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 823–852. Springer, Heidelberg, August 2020.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1054–1067, New York, NY, USA, 2014. ACM.

[GX17] Slawomir Goryczka and Li Xiong. A Comprehensive Comparison of Multiparty Secure Additions with Differential Privacy. *IEEE Transactions on Dependable and Secure Computing*, 14(5):463–477, 2017.

[HMM$^+$16] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, and Dan Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16. Association for Computing Machinery, 2016.

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1575–1590. ACM Press, November 2020.

[MPBB19] Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. In *Proceedings of the NeurIPS 2019 Workshop on Robust AI in Financial Services*, 2019.

[MS20] Ryan McKenna and Daniel Sheldon. Permute-and-Flip: A new mechanism for differentially private selection. *34th Conference on Neural Information Processing Systems (NeurIPS 2020),*, 33:193–203, 2020.

[MT07] F. McSherry and K. Talwar. Mechanism Design via Differential Privacy. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2007.

[Ste20] Thomas Steinke. Multi-Central Differential Privacy. arXiv preprint, 2020. arXiv:2009.05401.

[War65] Stanley L. Warner. Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

A Supplementary material

A.1 Sensitivity of Rounding

Our privacy analysis will need the property (2), repeated here for convenience:

$$\left|\text{round}_\Delta\left(\frac{x+\eta}{\gamma}\right) - \text{round}_\Delta\left(\frac{\bar{x}+\eta}{\gamma}\right)\right| \le 1 \ . \tag{2}$$

This bound follows from how the rounding function is implemeted in our MPC protocol. Note in particular that in this case we are not interested in the rounding error (i.e., the difference between the rounded value and the result of our approximate rounding function) but the sensitivity of the rounding function (i.e., the difference between the result of the approximate rounding function on two neighbouring inputs, regardless of their actual accuracy).

First remember that the users secret share x to the computing servers by picking $h-1$ uniformly random integers $x_1, \ldots, x_{h-1}$ from an appropriately large interval) and finally defining $x_h = x - \sum_{i\in[h-1]} x_i$ (resp. $\bar{x}_h = \bar{x} - \sum_{i\in[h-1]} x_i$), defining sharings $[x]_\mathbb{Z}$ and $[\bar{x}]_\mathbb{Z}$. Note that it is crucial that in this phase of the analysis we are fixing the randomness of both η and the random shares, and we are only varying the input. Now remember that the rounding function is being implemented by having each computing server locally rounding their value which leads to

$$\text{round}_\Delta\left(\frac{x+\eta}{\gamma}\right) = \sum_{i\in[h]} \lfloor (x_i+\eta)/\gamma \rceil \ .$$

Thus we get that

$$\begin{aligned}\left|\text{round}_\Delta\left(\frac{x+\eta}{\gamma}\right) - \text{round}_\Delta\left(\frac{\bar{x}+\eta}{\gamma}\right)\right| &= \left|\sum_{i\in[h]} \lfloor (x_i+\eta)/\gamma \rceil - \sum_{i\in[h]} \lfloor (\bar{x}_i+\eta)/\gamma \rceil\right| \\ &= |\lfloor (x_h+\eta)/\gamma \rceil - \lfloor (\bar{x}_h+\eta)/\gamma \rceil| \le 1\end{aligned}$$

Where the last inequality follows noticing that $x, \bar{x}$ are at most 1 apart.

A.2 Privacy Analysis

As a warm-up we analyze an easier special case, after which we handle the general case.

Lemma 4. *If $\Delta = 0$ and $\gamma = 1$, Algorithm 1 is ε-differentially private.*

Proof. Let $\mathcal{M}(\boldsymbol{x})$ denote the output of Algorithm 1 on input with sum $\boldsymbol{x} \in \mathbf{Z}^d$. Notice that $\mathcal{M}(\boldsymbol{x}) = i$ if and only if

$$\boldsymbol{x}_i + \boldsymbol{\eta}_i \ge \max_{i'\neq i}(\boldsymbol{x}_{i'} + \boldsymbol{\eta}_{i'} + [i' > i]), \tag{3}$$

where $[i' > i]$ equals 1 if the condition $i' > i$ holds and 0 otherwise. Consider a neighboring dataset with sum $\bar{\boldsymbol{x}}$. By definition of the neighboring relation it follows that both the left and right hand side of (3) change by at most 1 when replacing $\boldsymbol{x}$ with $\bar{\boldsymbol{x}}$. Using independence and the tail bound on the geometric distribution, we bound

$$\begin{aligned}\Pr[\mathcal{M}(\boldsymbol{x}) = i] &= \sum_y \Pr[\max_{i'\neq i}(\boldsymbol{x}_{i'} + \boldsymbol{\eta}_{i'} + [i' > i]) = y] \Pr[\boldsymbol{x}_i + \boldsymbol{\eta}_i \ge y] \\ &\le \sum_y \Pr[\max_{i'\neq i}(\boldsymbol{x}_{i'} + \boldsymbol{\eta}_{i'} + [i' > i]) = y] \Pr[\boldsymbol{x}_i + \boldsymbol{\eta}_i \ge y + 2]\, e^\varepsilon \\ &= e^\varepsilon \Pr[\boldsymbol{x}_i + \boldsymbol{\eta}_i \ge \max_{i'\neq i}(\boldsymbol{x}_{i'} + \boldsymbol{\eta}_{i'} + [i' > i]) + 2] \\ &\le e^\varepsilon \Pr[\bar{\boldsymbol{x}}_i + \boldsymbol{\eta}_i \ge \max_{i'\neq i}(\bar{\boldsymbol{x}}_{i'} + \boldsymbol{\eta}_{i'} + [i' > i])] \\ &= e^\varepsilon \Pr[\mathcal{M}(\bar{\boldsymbol{x}}) = i] \ .\end{aligned}$$

By symmetry we also have $\Pr[\mathcal{M}(\bar{\boldsymbol{x}}) = i] \le e^\varepsilon \Pr[\mathcal{M}(\boldsymbol{x}) = i]$, as desired. □

Table 2: Complexity Analysis, $k > 3$

	Bits sent	Rounds
Offline	$\mathcal{O}(da^2 k \log k)$	$\mathcal{O}(1)$
Online	$\mathcal{O}(akd)$	$\mathcal{O}(\log d \log a)$

Table 3: Complexity Analysis, $k = 3$

	Bits sent	Rounds
Offline	$\mathcal{O}(da^2)$	$\mathcal{O}(1)$
Online	$\mathcal{O}(ad)$	$\mathcal{O}(\log d \log a)$

We are ready to prove Lemma 1, which generalizes Lemma 4 to any value of the parameters:

Proof. The key difference to the proof of Lemma 4 is that while $\boldsymbol{x}_i + \boldsymbol{\eta}_i$ and $\bar{\boldsymbol{x}}_i + \boldsymbol{\eta}_i$ differ by at most 1, we now use (2) to bound $\Pr[\mathcal{M}(\boldsymbol{x}) = i]$ by

$$\begin{aligned}
&\sum_y \Pr\left[\max_{i' \neq i}\left(\text{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) + [i' > i]\right) = y\right] \Pr\left[\text{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) \geq y\right] \\
&\leq \sum_y \Pr\left[\max_{i' \neq i}\left(\text{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) + [i' > i]\right) = y\right] \Pr\left[\text{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) \geq y + 2\right] e^\varepsilon \\
&= e^\varepsilon \Pr\left[\text{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) \geq \max_{i' \neq i}\left(\text{round}_\Delta\left(\frac{\boldsymbol{x}_i + \boldsymbol{\eta}_i}{\gamma}\right) + [i' > i]\right) + 2\right] \\
&\leq e^\varepsilon \Pr\left[\text{round}_\Delta\left(\frac{\bar{\boldsymbol{x}}_i + \boldsymbol{\eta}_i}{\gamma}\right) \geq \max_{i' \neq i}\left(\text{round}_\Delta\left(\frac{\bar{\boldsymbol{x}}_i + \boldsymbol{\eta}_i}{\gamma}\right) + [i' > i]\right)\right] \\
&= e^\varepsilon \Pr[\mathcal{M}(\bar{x}) = i] \enspace .
\end{aligned}$$

By symmetry we also have $\Pr[\mathcal{M}(\bar{\boldsymbol{x}}) = i] \leq e^\varepsilon \Pr[\mathcal{M}(\boldsymbol{x}) = i]$, completing the proof. □

A.3 MPC Protocol Analysis

We offer an analysis of the complexity associated with the operations performed by the servers in Algorithm 4, in terms of the number of necessary communication rounds and the number of bits communicated during the protocol. The local generation of noise by each server and the generation of shares of this noise by supporting servers incur no communication. However, one round of communication is necessary in order for all supporting servers to distribute their noise shares to the computing servers. Adding the shared values and the noise vectors, as well as locally truncating the resulting shares and converting them to shares over a ring, require no communication. Since the argmax is clearly the bottleneck, we will analyze that.

In order to run the `ArgMax` protocol, the preprocessing step involves generating additive shares modulo 2^a for $a(d-1)$ random bits, because each of $d-1$ comparisons requires shares of a bits. Specifically in the case of 3 servers, the dealer can generate these shares locally, so only one round of communication to distribute the shares is necessary, and the number of bits to communicate will be $\mathcal{O}(da^2)$.

Preprocessing of each secret shared bit with more than 3 servers is done using the techniques from [ACD+19]. This involves generating a random shared value and a constant number of multiplications. This can be done while communicating $\mathcal{O}(k)$ elements of the ring over which the preprocessing is done. Due to the fact that we need "Shamir-style" secret sharing for the multiplications, we need to use a ring extension of $\mathbb{Z}_{2^a}$, where elements have size $a \log(k)$ bits, so we get communication of $\mathcal{O}(ak \log(k))$ bits per shared random bit and so a total of $\mathcal{O}(a^2 k \log(k))$ because we need a random shared bits. Since all these bits can be created in parallel, we can do them all in a constant number of rounds. We also need $\mathcal{O}(a)$ multiplication triples for multiplying bits, these can be done in the same complexity using the same techniques.

After precomputation is complete, running the `ArgMax` protocol requires $\mathcal{O}(d)$ comparisons in a circuit structure with depth $\mathcal{O}(\log d)$. Each comparison requires opening two secret shared values and executing two binary LT circuits. The LT circuit consists of $2a-2$ multiplications, including two share openings each, and can be done using a circuit of depth $\log a$, where the depth indicates the number of necessary rounds. Therefore, this step incurs $\mathcal{O}(ad)$ share openings and multiplications, and $\mathcal{O}(\log a \log d)$ rounds of communication. Since k servers are involved, these share openings and multiplications require communication $\mathcal{O}(akd)$, which is $\mathcal{O}(ad)$ if $k=3$.

In total, the total communication and number of rounds when $k>3$ is summarized in Table 2 and when $k=3$ is summarized in Table 3.

A.4 Utility evaluation

Here we present an individual plot for running the algorithms on each of the five datasets from DPBench, as well as the mean error and standard deviation in tabular format. We pick the values of ε to be as small as possible to capture when the most of the algorithms converge to an error of 0. Note that such small values for ε creates a high standard deviation for some datasets, e.g., for ADULTFRANK (see Table 8). The full outputs from the experiments are documented in the accompanying csv files in the `code/results` folder.

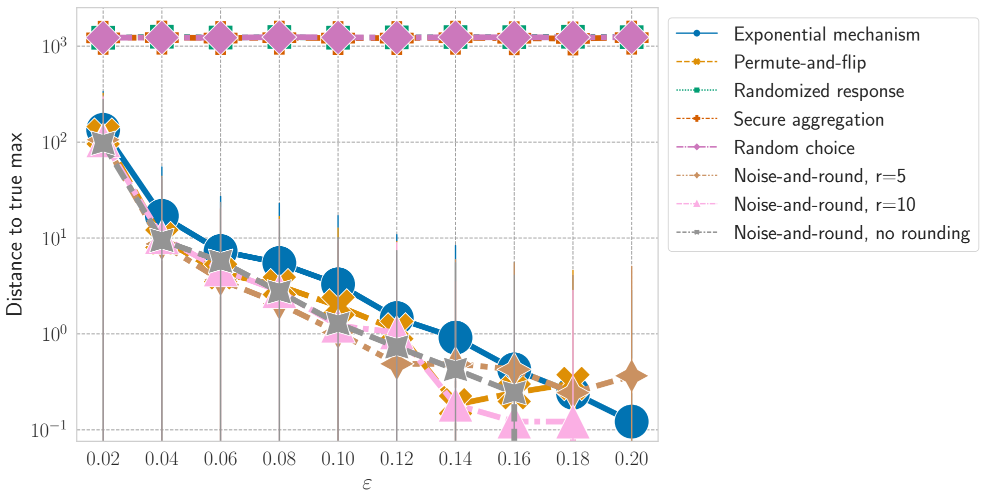


Figure 3: HEPTH dataset. Absolute difference between the real chosen argmax on a log-log scale. Lower is better.

Table 4: HEPTH summary, Noise-and-round without rounding

	Dataset	**Max value**	**Bits**
	HEPTH	1571	11
Algorithm	ε	**Mean error**	**Standard deviation**
Permute-and-flip	0.02	117.758	204.729
Noise-and-round	0.02	97.247	181.122
Permute-and-flip	0.04	9.829	31.0666
Noise-and-round	0.04	9.444	33.634
Permute-and-flip	0.06	4.331	15.674
Noise-and-round	0.06	5.734	17.810
Permute-and-flip	0.08	3.172	13.550
Noise-and-round	0.08	2.745	12.652
Permute-and-flip	0.10	1.952	10.7414
Noise-and-round	0.10	1.281	8.751
Permute-and-flip	0.12	1.098	8.114
Noise-and-round	0.12	0.732	6.645
Permute-and-flip	0.14	0.183	3.338
Noise-and-round	0.14	0.427	5.088
Permute-and-flip	0.16	0.244	3.852
Noise-and-round	0.16	0.244	3.852
Permute-and-flip	0.18	0.305	4.305
Noise-and-round	0.18	0.0	0.0
Permute-and-flip	0.20	0.0	0.0
Noise-and-round	0.20	0.0	0.0

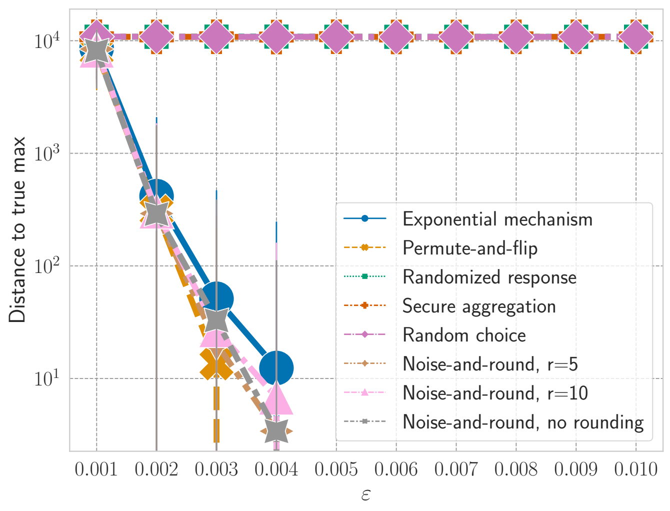


Figure 4: SEARCHLOGS dataset. Absolute difference between the real chosen argmax on a log-log scale. Lower is better.

Table 5: SEARCHLOGS summary, Noise-and-round without rounding

	Dataset	**Max value**	**Bits**
	SEARCHLOGS	11160	14
Algorithm	ε	**Mean error**	**Standard deviation**
Permute-and-flip	0.001	8039.062	4354.840
Noise-and-round	0.001	8145.286	4347.058
Permute-and-flip	0.002	303.066	1429.518
Noise-and-round	0.002	288.855	1472.448
Permute-and-flip	0.003	13.616	214.964
Noise-and-round	0.003	32.812	350.543
Permute-and-flip	0.004	0.0	0.0
Noise-and-round	0.004	3.404	107.644
Permute-and-flip	0.005	0.0	0.0
Noise-and-round	0.005	0.0	0.0
Permute-and-flip	0.006	0.0	0.0
Noise-and-round	0.006	0.0	0.0
Permute-and-flip	0.007	0.0	0.0
Noise-and-round	0.007	0.0	0.0
Permute-and-flip	0.008	0.0	0.0
Noise-and-round	0.008	0.0	0.0
Permute-and-flip	0.009	0.0	0.0
Noise-and-round	0.009	0.0	0.0
Permute-and-flip	0.010	0.0	0.0
Noise-and-round	0.010	0.0	0.0

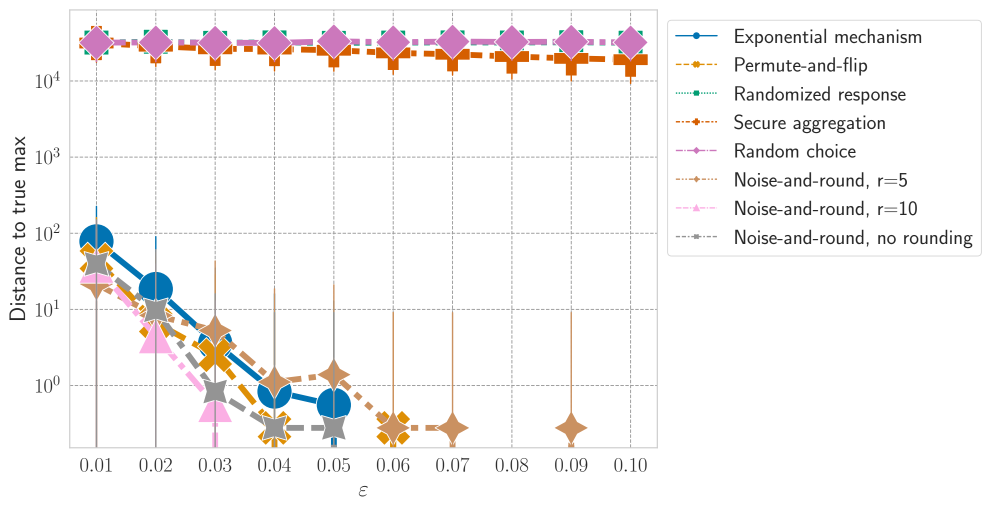


Figure 5: PATENT dataset. Absolute difference between the real chosen argmax on a log-log scale. Lower is better.

Table 6: PATENT summary, Noise-and-round without rounding

	Dataset	**Max value**	**Bits**
	PATENT	59602	16
Algorithm	ε	**Mean error**	**Standard deviation**
Permute-and-flip	0.01	44.862	115.574
Noise-and-round	0.01	39.58	111.060
Permute-and-flip	0.02	6.672	42.569
Noise-and-round	0.02	9.73	51.116
Permute-and-flip	0.03	2.502	26.268
Noise-and-round	0.03	0.834	15.211
Permute-and-flip	0.04	0.278	8.791
Noise-and-round	0.04	0.278	8.791
Permute-and-flip	0.05	0.0	0.0
Noise-and-round	0.05	0.278	8.791
Permute-and-flip	0.06	0.278	8.791
Noise-and-round	0.06	0.0	0.0
Permute-and-flip	0.07	0.0	0.0
Noise-and-round	0.07	0.0	0.0
Permute-and-flip	0.08	0.0	0.0
Noise-and-round	0.08	0.0	0.0
Permute-and-flip	0.09	0.0	0.0
Noise-and-round	0.09	0.0	0.0
Permute-and-flip	0.10	0.0	0.0
Noise-and-round	0.10	0.0	0.0

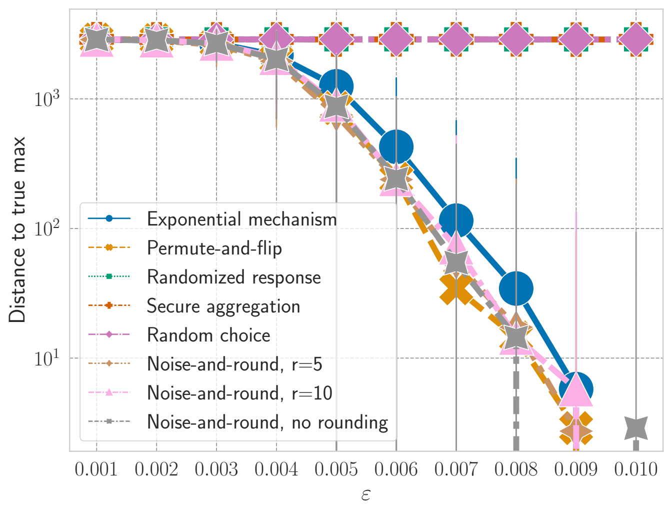


Figure 6: MEDCOST dataset. Absolute difference between the real chosen argmax on a log-log scale. Lower is better.

Table 7: MEDCOST summary, Noise-and-round without rounding

	Dataset	**Max value**	**Bits**
	MEDCOST	2885	12
Algorithm	ε	**Mean error**	**Standard deviation**
Permute-and-flip	0.001	2866.961	184.046
Noise-and-round	0.001	2865.654	184.471
Permute-and-flip	0.002	2839.364	339.219
Noise-and-round	0.002	2831.078	362.647
Permute-and-flip	0.003	2690.079	710.319
Noise-and-round	0.003	2671.08	739.789
Permute-and-flip	0.004	2020.671	1317.405
Noise-and-round	0.004	2016.477	1317.891
Permute-and-flip	0.005	847.703	1311.318
Noise-and-round	0.005	874.724	1324.313
Permute-and-flip	0.006	240.803	795.751
Noise-and-round	0.006	241.873	799.149
Permute-and-flip	0.007	34.55	313.660
Noise-and-round	0.007	54.5	391.862
Permute-and-flip	0.008	14.423	203.563
Noise-and-round	0.008	14.422	203.549
Permute-and-flip	0.009	2.885	91.232
Noise-and-round	0.009	0.0	0.0
Permute-and-flip	0.010	0.0	0.0
Noise-and-round	0.010	2.884	91.200

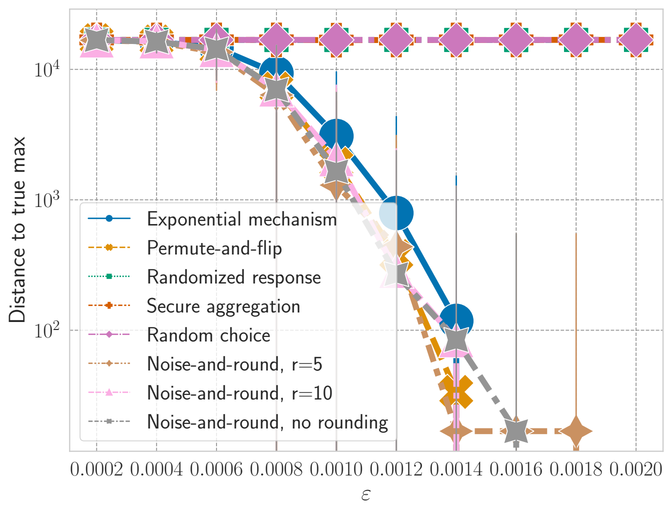


Figure 7: ADULTFRANK dataset. Absolute difference between the real chosen argmax on a log-log scale. Lower is better.

Table 8: ADULTFRANK summary, Noise-and-round without rounding

	Dataset	Max value	Bits
	ADULTFRANK	16836	15
Algorithm	ε	**Mean error**	**Standard deviation**
Permute-and-flip	0.0002	16734.164	1300.793
Noise-and-round	0.0002	16801.488	752.541
Permute-and-flip	0.0004	16380.53	2730.059
Noise-and-round	0.0004	16313.562	2919.344
Permute-and-flip	0.0006	14007.064	6297.344
Noise-and-round	0.0006	14293.236	6030.906
Permute-and-flip	0.0008	7121.219	8321.266
Noise-and-round	0.0008	7003.551	8302.241
Permute-and-flip	0.0010	1885.368	5311.424
Noise-and-round	0.0010	1649.898	5008.003
Permute-and-flip	0.0012	370.387	2470.763
Noise-and-round	0.0012	269.376	2113.556
Permute-and-flip	0.0014	33.672	752.552
Noise-and-round	0.0014	84.18	1188.099
Permute-and-flip	0.0016	0.0	0.0
Noise-and-round	0.0016	16.836	532.401
Permute-and-flip	0.0018	0.0	0.0
Noise-and-round	0.0018	0.0	0.0
Permute-and-flip	0.0020	0.0	0.0
Noise-and-round	0.0020	0.0	0.0

A.5 Efficiency evaluation

All efficiency results for the five chosen datasets from DPBench are reported in Table 9, including the maximum value in each dataset, the number of bits necessary to represent integers in this range, as well as the runtimes and data sent. The five datasets are the same datasets chosen for evaluation by [MS20] in the Permute-and-flip mechanism: PATENT, ADULTFRANK, SEARCHLOGS, MEDCOST, and HEPTH.

Table 9: Benchmark results for given input

			Results	
Dataset	Max value (n)	# Bits	Time (s)	Data sent (MB)
PATENT	59602	16	3.83, std=0.18	2.75
ADULTFRANK	16836	15	4.21, std=0.14	2.61
SEARCHLOGS	11160	14	3.79, std=0.14	2.48
MEDCOST	2885	12	4.15, std=0.33	2.20
HEPTH	1571	11	3.80, std=0.11	2.07
Truncated, $\alpha = 0.125$	31	5	2.80, std=0.10	1.78