Secure Multi-party Computation of Differentially Private Median

Jonas Böhler, *SAP Security Research;* Florian Kerschbaum, *University of Waterloo*

https://www.usenix.org/conference/usenixsecurity20/presentation/boehler

This paper is included in the Proceedings of the 29th USENIX Security Symposium.
August 12–14, 2020
978-1-939133-17-5

Secure Multi-party Computation of Differentially Private Median

Jonas Böhler
SAP Security Research

Florian Kerschbaum
University of Waterloo

Abstract

In this work, we consider distributed private learning. For this purpose, companies collect statistics about telemetry, usage and frequent settings from their users without disclosing individual values. We focus on rank-based statistics, specifically, the median which is more robust to outliers than the mean.

Local differential privacy, where each user shares locally perturbed data with an untrusted server, is often used in private learning but does not provide the same accuracy as the central model, where noise is applied only once by a trusted server. Existing solutions to compute the differentially private median provide good accuracy only for large amounts of users (local model), by using a trusted third party (central model), or for a very small data universe (secure multi-party computation).

We present a multi-party computation to efficiently compute the exponential mechanism for the median, which also supports, e.g., general rank-based statistics (e.g., p^{th}-percentile, interquartile range) and convex optimizations for machine learning. Our approach is efficient (practical running time), scaleable (sublinear in the data universe size) and accurate, i.e., the absolute error is smaller than comparable methods and is independent of the number of users, hence, our protocols can be used even for a small number of users. In our experiments we were able to compute the differentially private median for 1 million users in 3 minutes using 3 semi-honest computation parties distributed over the Internet.

1 Introduction

We consider the problem of distributed private learning. Specifically, how multiple users can compute rank-based statistics over their sensitive data, with high accuracy, a strong privacy guarantee, and without resorting to trusted third parties. Rank-based statistics include the median, p^{th}-percentiles, and interquartile ranges, and we present a protocol to compute the differentially private median, which is extensible to any k^{th} ranked element. We use *differential privacy* (DP) [25, 28], a rigorous privacy notion, restricting what can be inferred about any individual in the data, used by Google [15, 31], Apple [1, 66], Microsoft [23] and the US Census bureau [2]. The median is a robust statistical method used to represent a "typical" value from a data set, e.g., insurance companies use the median life expectancy to adjust insurance premiums.

Previous work on DP median computation either require a large number of users to be accurate [27, 34, 63], rely on a trusted third party [51, 58], or cannot scale to large universe or data set sizes [14, 30, 59]. We present a novel alternative that is superior in accuracy, requires no trusted party, and is efficiently computable. Our protocol provides high accuracy even for a small number of users. Note that small sample size is the most challenging regime for DP [56]. Even Google's large-scale data collection (billions of daily reports via [31]) is insufficient if the statistical value of interest is not a heavy hitter [15], e.g., the median.

We present a *secure multi-party computation* (MPC) of the *exponential mechanism* [52] for *decomposable aggregate functions*. Such functions, as used in MapReduce-style algorithms [22], allow efficient aggregation in parallel over distributed data sets, and application examples include convex loss functions and rank-based statistics. The exponential mechanism can implement any differentially private algorithm by computing selection probabilities for all possible output elements. Its computation complexity is linear in the size of the data universe [52] and efficiently sampling it is non-trivial [29]. Also, the exponential mechanism requires exponentiations, increasing the MPC complexity. However, as it is a universal mechanism, a scalable, secure implementation can be widely applied. Eigner et al. [30] also implement the exponential mechanism in MPC. They compute the exponential function with MPC, whereas we provide a more efficient alternative for decomposable functions. Their approach, while more general, is only practical for a universe size of 5 elements, whereas our protocol is sublinear in the size of the universe and handles billions of elements. We achieve this via divide-and-conquer and optimizing our protocol for decomposable functions that enable efficient alternatives to expensive secure computation of exponentiations [5,7,20,43].

In summary, our contribution is a protocol for securely computing the differentially private median

- with high accuracy even for small data sets (few users) and large universe sizes (see Section 3.4 for our theoretical errors bounds, Appendix F for a comparison of that bound to related work, and Section 5.3 for empirical comparison to related work),
- that is efficient (practical running time for millions of users) and scalable (sublinear in the data universe size) (Sections 4, 5),
- secure in the semi-honest model with an extension to the malicious model (Section 4.6) and outputs the differentially private median according to the exponential mechanism by McSherry and Talwar [52],
- evaluated using an implementation in the SCALE-MAMBA framework [6], for 1 million users using 3 semi-honest computation parties with a running time of seconds in a LAN, and 3 minutes in a WAN with 100 ms network delay, 100 Mbits/s bandwidth (Section 5).

The remainder of this paper is organized as follows: In Section 2 we describe preliminaries for our protocol. In Section 3 we explain our protocol and introduce definitions. We present our protocol and implementation details for the secure multi-party computation of the differentially private median in Section 4. We provide a detailed performance evaluation in Section 5, describe related work in Section 6 and conclude in Section 7.

2 Preliminaries

In the following, we introduce preliminaries for differential privacy and secure multi-party computation.

We consider a set of input parties $\mathcal{P} = \{P_1, \dots, P_n\}$, where party P_i holds a datum d_i, and D denotes their combined data set. We model a data set as $D = \{d_1, \dots, d_n\} \in U^n$ with underlying *data universe* U. We also consider m semi-honest *computation parties*, e.g., $m \in \{3, 6, 10\}$, who run the computation on behalf of the input parties. To simplify presentation, we assume the size n of D to be even, which can be ensured by padding. Then, the median's position in sorted D is $n/2$.

2.1 Differential Privacy

Differential privacy (DP), introduced by Dwork et al. [25,28], is a strong privacy guarantee restricting what a mechanism operating on a sensitive data set can output. Informally, when the input data set changes in a single element, the effect on the output is bounded. The formal definition is as follows:

Definition 1 (Differential Privacy)**.** *A mechanism $\mathcal{M}$ satisfies ε-differential privacy, where $\varepsilon \geq 0$, if for all neighboring data sets $D \simeq D'$, i.e., data sets differing in a single entry, and all sets $S \subseteq Range(\mathcal{M})$*

$$Pr[\mathcal{M}(D) \in S] \leq \exp(\varepsilon) \cdot Pr\left[\mathcal{M}(D') \in S\right],$$

where $Range(\mathcal{M})$ denotes the set of all possible outputs of mechanism $\mathcal{M}$.

The above definition holds against an unbounded adversary, however, due to our use of cryptography we assume a computationally bounded adversary. A formal definition is presented in Appendix A based on MPC preliminaries from Section 2.2.

Randomization is essential for differential privacy to hide an individual's inclusion in the data [29]. Noise, added to the function output, is one way to achieve differential privacy, e.g., via the *Laplace mechanism* [29]:

Definition 2 (Laplace Mechanism)**.** *Given a function $f : U^n \to \mathbb{R}$ with sensitivity $\max_{\forall D \simeq D'} |f(D) - f(D')|$, privacy parameter ε, and a database D, the* Laplace mechanism *releases $f(D) + r$, where r is drawn from the Laplace distribution (centered at 0) with density $\frac{\varepsilon}{2\Delta f} e^{\frac{-\varepsilon}{\Delta f}}$.*

The alternative to additive noise is probabilistic output selection via the *exponential mechanism*, introduced by McSherry and Talwar [52]. The exponential mechanism expands the application of differential privacy to functions with non-numerical output, or when the output is not robust to additive noise, e.g., the median function [48]. The mechanism is exponentially more likely to select "good" results where "good" is quantified via a utility function $u(D, r)$ which takes as input a database $D \in U^n$, and a potential output $r \in \mathcal{R}$ from a fixed set of arbitrary outputs $\mathcal{R}$. Informally, higher utility means the output is more desirable and its selection probability is increased accordingly.

Definition 3 (Exponential Mechanism)**.** *For any utility function $u : (U^n \times \mathcal{R}) \to \mathbb{R}$ and a privacy parameter ε, the exponential mechanism $\mathsf{EM}_u^\varepsilon(D)$ outputs $r \in \mathcal{R}$ with probability proportional to $\exp(\frac{\varepsilon u(D,r)}{2\Delta u})$, where*

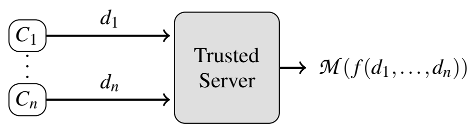
$$\Delta u = \max_{\forall r \in \mathcal{R}, D \simeq D'} \left| u(D, r) - u\left(D', r\right) \right|$$

is the sensitivity of the utility function. That is,

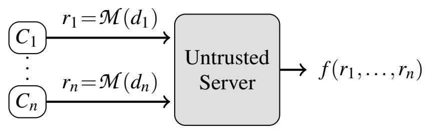
$$Pr[\mathsf{EM}_u^\varepsilon(D) = r] = \frac{\exp\left(\frac{\varepsilon u(D,r)}{2\Delta u}\right)}{\sum_{r' \in \mathcal{R}} \exp\left(\frac{\varepsilon u(D,r')}{2\Delta u}\right)}. \quad (1)$$

We omit u, ε, D, i.e., write EM, if they can be derived from the context.

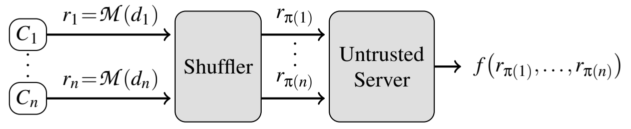
DP algorithms $\mathcal{M}$ can be implemented in different models, visualized in Figure 1. Next, we describe the models and explain which model we implement.



(a) Central Model



(b) Local Model

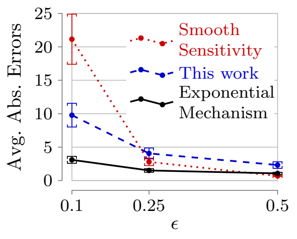


(c) Shuffle Model with permutation π

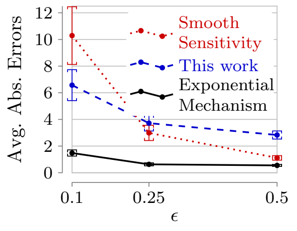
Figure 1: Models for DP mechanism $\mathcal{M}$. Client C_i sends a message (raw data d_i or randomized r_i) to a server, who computes function f over the messages, and releases the result.

2.1.1 Why We Consider the Central Model

In the *central model* (Figure 1a) every client sends their unprotected data to a trusted, central server which runs $\mathcal{M}$ on the clear data. The central model provides the highest accuracy as the randomization inherent to DP algorithms, is only applied once. In the *local model* (Figure 1b), introduced by [44], clients apply $\mathcal{M}$ locally and sent anonymized values to an untrusted server for aggregation. The accuracy is limited as the randomization is applied multiple times. Hence, it requires a very large number of users to achieve accuracy comparable to the central model [15, 18, 40, 44, 50]. Specifically, an exponential separation between local and central model for accuracy and sample complexity was shown by [44]. Recently, an intermediate *shuffle model* (Figure 1c) was introduced [15, 18]: A trusted party is added between client and server in the local model, the shuffler, who does not collude with anyone. The shuffler permutes and forwards the randomized client values. The permutation breaks the mapping between a client and her value, which reduces randomization requirements. The accuracy of the shuffle model lies between the local and central model, however, in general it is strictly weaker than the central model [9, 18]. As our goal is high accuracy without trusted parties even for small number of users, we *simulate* the central model in a distributed setting via secure multi-party computation (MPC), which is often used in DP literature [26, 30, 38, 59, 60, 65]. MPC, further described in Section 2.2, is a cryptographic protocol run by clients over their sensitive data that only reveals the computation output without requiring a trusted server. General MPC incurs high computation and communication overhead which reduce efficiency and scalability [18]. However, MPC combines the respective benefits of the models, namely, high accuracy and strong privacy, i.e., no disclosure of values to a third party, and we present an efficient and scaleable MPC protocol.



(a) Credit card transactions [67], first 10^5 payment records in Cents.



(b) Walmart supply chain data [42], 175k shipment weights as integers.

Figure 2: Absolute errors, averaged for 100 differentially private median computations via Laplace mechanism with smooth sensitivity, this work, and the exponential mechanism.

2.1.2 Why We Use the Exponential Mechanism

Next, we illustrate why the exponential mechanism offers better accuracy than additive noise w.r.t. the DP median. Recall, the noise depends on the sensitivity of function f and the privacy parameter ε. The sensitivity is the largest difference a single change in *any possible database* can have on the function result. *Smooth sensitivity*, developed by Nissim et al. [58], additionally analyzes the data to provide instance-specific additive noise that is often much smaller. (See Appendix F for a formal description.) However, computation of smooth sensitivity requires access to the entire data set, otherwise the error increases further[1], which prohibits efficient (secure) computation with high accuracy. Li et al. [48] note that the Laplace mechanism is ineffective for the median as (smooth) sensitivity can be high. Additionally, they present a median utility function for the exponential mechanism with low, *data-independent* sensitivity, which we use in our protocol. To illustrate that additive noise can be high, we empirically evaluated the absolute error of the Laplace mechanism with smooth sensitivity, the exponential mechanism, and our protocol in Figure 2 on real-world data sets [42, 67]. Our protocol uses the exponential mechanism in multiple steps, and while the accuracy is not the same as for (single use of) the exponential mechanism, we do not require a trusted third party. Overall, we achieve better accuracy than additive noise for low ε (corresponding to high privacy protection) with better scalability than the exponential mechanism. We provide our accuracy bounds in Section 3.4, further empirical evaluations w.r.t. related work in Section 5.3, and describe related work in Section 6.

[1]Smooth sensitivity approximations exist that provide a factor of 2 approximation in linear-time, or an additive error of $\max(U)/\text{poly}(|D|)$ in sublinear-time [58, Section 3.1.1]. Note that this error e is w.r.t. smooth sensitivity s, the additive noise is even larger with Laplace$((s+e)/\varepsilon)$.

2.2 Secure Multi-party Computation

Secure multi-party computation (MPC) [36] allows a set of three or more parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where party P_i holds sensitive input d_i, to jointly compute a function $y = f(d_1, \ldots, d_n)$ while protecting their inputs. The computation must be *correct*, i.e., the correct y is computed, and *secret*, i.e., only y and nothing else is revealed. There are two main implementation paradigms for MPC [32,46]: *garbled circuits* [68][2], where the parties construct a (large, encrypted) circuit and evaluate it at once, and *secret sharing* [12, 21, 57, 62], where the parties interact for each circuit gate. In general, the former allows for constant number of rounds but requires larger bandwidth (as fewer, but bigger messages are sent), and the latter has low bandwidth (small messages per gate) and high throughput, where the number of rounds depends on the circuit depth. We will focus on secret-sharing-based MPC as our goal is an efficient implementation in a network with reasonable latency. Informally, a (t,n)-secret sharing scheme splits a secret s into n shares s_i and at least t shares are required to reconstruct the secret. We use $\langle s \rangle = (s_1, \ldots, s_n)$ to denote the sharing of s among n parties (for a formal definition see, e.g., Evans et al. [32]). Recent works, e.g., SCALE-MAMBA [6], BDOZ [12], SPDZ [21], improve MPC performance by combining a computationally secure *offline phase*, to exchange correlated randomness (e.g., Beaver triples [11]), with an information-theoretic secure *online phase*. The former is generally more efficient since the latter requires asymmetric cryptography [47]. MPC can be implemented in two models with different trust assumptions: in the *semi-honest model* (passive) adversaries do not deviate from the protocol but gather everything created during the run of the protocol, in the *malicious model* (active) adversaries can deviate from the protocol (e.g., alter messages).

In this work we consider n *input parties* with sensitive input, and m (e.g., $m \in \{3, 6, 10\}$) semi-honest *computation parties*, i.e., non-colluding untrusted servers. The input parties create and send shares of their input to the computation parties, which run the secure computation on their behalf. We assume semi-honest parties but explain how to extend our protocol to malicious parties and implement our protocol with the SCALE-MAMBA framework [6].

3 Secure EM for Median Selection

We implement a multi-party computation of the exponential mechanism EM for rank-based statistics enabling distributed parties to learn the differentially private median of their joint data. There are two challenges for multi-party computation of the exponential mechanism:

[2] Yao described a garbled circuit for two parties in an oral presentation about secure function evaluation [68], the first written description is from [37], and the first proof was given in [49].

(i) the running time complexity is linear in the size of the data universe, $|U|$, as selection probabilities for *all* possible outputs in U are computed,

(ii) the general mechanism is too inefficient for general secure computation as selection probability computation requires $|U|$ exponentiations over floating-point numbers.

We solve these challenges by (i) recursively dividing the data universe into subranges to achieve sublinear running time in $|U|$, and (ii) focusing on utility functions which allow efficient selection probability computation. We call such utility functions *decomposable*, which we formalize in Section 3.1, and give example applications.

In the following, we describe an overview of our solution. We efficiently compute the exponential mechanism with running time complexity sublinear in the size of the data universe U by dividing U into k subranges. We select the best subrange and also split it into k subranges for the next iteration, until the last subrange is small enough to directly select the final output from it. After $\lceil \log_k |U| \rceil$ iterations the selected subrange contains only one element. Each subrange selection increases the overall privacy loss ε, and we enable users to select a trade-off between running time, privacy loss and accuracy by presenting three protocols to compute unnormalized selection probabilities, which we call *weights*, w.r.t. ε:

- Weights$^{\ln(2)}$ fixes $\varepsilon = \ln(2)$ to compute $\exp(\varepsilon y)$ as 2^y,
- Weights$^{\ln(2)/2^d}$ allows $\varepsilon = \frac{\ln(2)}{2^d}$ for some integer $d > 0$,
- Weights* supports arbitrary ε.

On a high-level, we have three phases in each iteration:

1. *Evaluate*: Each party locally computes the basis for utility scores for each subrange.
2. *Combine*: They combine their results into a global result and compute selection probabilities.
3. *Select*: Finally, they select an output based on its selection probabilities.

The results of the evaluation step are computed over sensitive data and might also be sensitive (e.g., utility functions for median and mode leak exact counts [48]). Therefore, we combine them via MPC to preserve privacy. To ensure efficient implementation of the combination step we require utility functions to have a certain structure as detailed next.

3.1 Decomposability & Applications

Recall, each party P_i holds a single value d_i (we can generalize to data sets D_i). To combine local utility scores per party into a global score for all, we require utility functions to be *decomposable*:

Application	*Utility*
Convex optimization: find x that minimizes $\sum_{i=1}^{n} l(x,d_i)$ with convex loss function l defined over D; e.g., empirical risk minimization in machine learning [10, 63], and integer partitions (password frequency lists) [16]	$-\sum_{i=1}^{n} l(x,d_i)$
Unlimited supply auction: find price x maximizing revenue $x\sum_i b_i(x)$, where bidder demand curve b_i indicates how many goods bidder i will buy at price x; e.g., digital goods [52]	$x\sum_i b_i(x)$
Frequency: select x based on its frequency in D; e.g., mode [48]	$\sum_{i=1}^{n} \mathbb{1}_{x=d_i}$
Rank-based statistics: select x based on its rank in sorted D; e.g., k^{th}-ranked element [48]	See Section 3.2

Table 1: Applications with *decomposable* utility functions.

Definition 4 (Decomposability). *We call a function* $u : (U^n \times \mathcal{R}) \to \mathbb{R}$ decomposable *w.r.t. function* $u' : (U^n \times \mathcal{R}) \to \mathbb{R}$ *if* $u(D,x) = \sum_{i=1}^{n} u'(d_i,x)$ *for* $x \in \mathcal{R}$ *and* $D = \{d_1,\dots,d_n\}$.

We use decomposability to easily combine utility scores in $\text{Weights}^{\ln(2)}$, $\text{Weights}^{\ln(2)/2^d}$, and to avoid secure evaluation of the exponential function in Weights^{*}[3]. If u is decomposable, users can compute weights locally, and securely combine them via multiplications:

$$\prod_i \exp(u'(d_i,x)\varepsilon) = \exp(\sum_i u'(d_i,x)\varepsilon) = \exp(u(D,x)\varepsilon).$$

Decomposability is satisfied by a wide range of selection problems. Counts are clearly decomposable and so are utility functions that can be expressed as a sum of utility scores. Applications with decomposable utility functions are listed in Table 1. One use case for the median is a software company collecting private usage statistics, e.g., number of times a procedure was run or the size of database tables, in a medium-sized installed base. Reporting the median in addition to the mean allows the collector to detect skew in the distribution. Another example is private federated learning with network resource constrained parties, e.g., mobile phones on cellular networks. Gradient compressed federated learning, e.g., signSGD [13], enables to reduce the update message size for these parties, but uses the median instead of the mean to aggregate the gradients. The additional communication stemming from our secure median computation can be shifted to few parties who are not network resource constrained, e.g., mobile phones on WiFi networks.

To be sublinear in the size of the universe we consider decomposability w.r.t. ranges instead of elements: parties only report one utility score per range, instead of one score per element. Decomposability for elements $x \in U$ does not imply decomposability for ranges $R \subset U$[4]. However, we present a decomposable utility function w.r.t. ranges for rank-based statistics next.

[3]Secure exponentiation is complex [5,7,20,43], requiring many interactive rounds, and we want to avoid the expensive computational overhead.

3.2 Decomposable Median Utility Function

First, we describe the median utility function [48]. Then, we present a reformulation more convenient for secure implementation and show that it is decomposable.

Li et al. [48, Section 2.4.3] quantify an element's utility via its *rank* relative to the median. The rank of $x \in U$ in a data set D is the number of values in D smaller than x. More formally, $\text{rank}_D(x) = |\{d \mid d \in D : d < x\}|$. Note that for the median we have $\mathcal{R} = U$, which means every universe element is a potential output. As U can be large, we divide U in k equal-sized ranges, and define utility per range next.

Definition 5 (Median Utility Function). *The* median utility function $u_\mu : (U^n \times U) \to \mathbb{Z}$ *gives a utility score for a range* $R = [r_l, r_u)$ *where* $r_l, r_u \in U$ *w.r.t.* $D \in U^n$ *as*

$$u_\mu(D,R) = - \min_{\text{rank}_D(r_l) \le j \le \text{rank}_D(r_u)} \left| j - \frac{n}{2} \right|.$$

We focus on MPC of the differentially private median with rank $n/2$ but Definition 5 supports any k^{th}-ranked element. The sensitivity of u_μ is $1/2$ since adding an element increases $n/2$ by $1/2$ and j either increases by 1 or remains the same [48]. Thus, the denominator $2\Delta u$ in the exponents of (1) equals 1, and we will omit it in the rest of this work.

To compute u_μ one needs to find rank j minimizing the distance between the median and all range elements by iterating over all j where $\text{rank}_D(r_l) \le j \le \text{rank}_D(r_u)$. However, a naive implementation of u_μ leaks information as the iteration count depends on the number of duplicates in the data. We adapt u_μ next to remove this leakage. To avoid iterating over range elements observe that the utility for a range $R = [r_l, r_u)$ is defined by the element in the range closest to the median μ. Thus, it suffices to consider three cases: The range is either positioned "before" the median ($r_u \le \mu$), contains it, or comes "after" it ($r_l > \mu$). This observation leads us to the following definition without iterations:

Definition 6 (Simplified Median Utility Function). *The* median utility function $u_\mu^c : (U^n \times U) \to \mathbb{Z}$ *gives a utility score for a range* $R = [r_l, r_u)$ *of* U *w.r.t.* $D \in U^n$ *as*

$$u_\mu^c(D,R) = \begin{cases} \text{rank}_D(r_u) - \frac{n}{2} & \text{if } \text{rank}_D(r_u) < \frac{n}{2} \\ \frac{n}{2} - \text{rank}_D(r_l) & \text{if } \text{rank}_D(r_l) > \frac{n}{2} \\ 0 & \text{else} \end{cases}.$$

[4]Consider the mode, i.e., the most frequent element. E.g., for two parties with data sets $D_1 = \{1,1,1,2,2\}, D_2 = \{2,2,3,3,3\}$ the mode per data set is 1 resp. 3 but the mode for the combined data is 2.

1. Set $s = \lceil \log_k |U| \rceil$ and split privacy budget ε into $\varepsilon_1, \dots, \varepsilon_s$
2. Initialize $S = U$ and repeat below steps s times:
   (a) Every party $p \in \mathcal{P}$ divides S into k equal-sized subranges $\{R^i = [r_l^i, r_u^i)\}_{i=1}^k$
      i. if $\varepsilon_j = \ln(2)/2^d$ in step j (with integer $d \geq 0$), input $\left\{\text{rank}_{D_p}(r_l^i), \text{rank}_{D_p}(r_u^i)\right\}_{i=1}^k, d$
      ii. else input $\left\{e^{\varepsilon_j\left(\text{rank}_{D_p}(r_u^i) - |D_p|/2\right)}, e^{\varepsilon_j\left(|D_p|/2 - \text{rank}_{D_p}(r_l^i)\right)}\right\}_{i=1}^k, \varepsilon_j$
   (b) The functionality combines the inputs (Section 3.2) and outputs $S = R^i$ with probability proportional to $\exp(u_\mu^c(D, R^i)\varepsilon_j)$

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{EM}^*}$ for EM^*.

In the following, we generalize from a single value per (input) party, d_i, to multiple values, i.e., data set D_i, as computation parties operate on data sets later on. Definition 5 and 6 are equivalent as can be seen by proof by cases (see Appendix B), and u_μ^c is decomposable w.r.t.:

$$u'(D_i, R) = \begin{cases} \text{rank}_{D_i}(r_u) - \frac{|D_i|}{2} & \text{if } \text{rank}_D(r_u) < \frac{n}{2} \\ \frac{|D_i|}{2} - \text{rank}_{D_i}(r_l) & \text{if } \text{rank}_D(r_l) > \frac{n}{2} \\ 0 & \text{else} \end{cases},$$

where $\text{rank}_D(r) = \sum_{i=1}^n \text{rank}_{D_i}(r)$ for range endpoints r. We will use both utility definitions interchangeably. Specifically, we use u_μ to simplify notation in our accuracy proofs (Section 3.4), and u_μ^c in our implementation (Section 4).

For implementations $\mathsf{Weights}^{\ln(2)}$, $\mathsf{Weights}^{\ln(2)/2^d}$ the parties input *ranks* for lower and upper range endpoints (as in u' above), which we combine (as u_μ^c) to efficiently compute weights. For $\mathsf{Weights}^*$ we let the parties input *weights*, i.e., $\exp(\varepsilon u')$, which we can efficiently combine via multiplication. In more detail, weights for u' are:

$$e^{\varepsilon \cdot u'(D_i, R)} = \begin{cases} e^{\varepsilon\left(\text{rank}_{D_i}(r_u) - \frac{|D_i|}{2}\right)} & \text{if } e^{\varepsilon\left(\text{rank}_D(r_u) - \frac{n}{2}\right)} < 1 \\ e^{\varepsilon\left(\frac{|D_i|}{2} - \text{rank}_{D_i}(r_l)\right)} & \text{if } 1 > e^{\varepsilon\left(\frac{n}{2} - \text{rank}_D(r_l)\right)} \\ 1 & \text{else} \end{cases},$$

where, e.g., $e^{\varepsilon\left(\text{rank}_D(r) - \frac{n}{2}\right)} = \prod_{i=1}^n e^{\varepsilon\left(\text{rank}_{D_i}(r) - \frac{|D_i|}{2}\right)}$ for range endpoints r. Given these inputs, we are ready to describe an idealized version of our protocol next.

3.3 Ideal Functionality $\mathcal{F}_{\mathsf{EM}^*}$

The ideal functionality $\mathcal{F}_{\mathsf{EM}^*}$ in Figure 3 describes our DP median protocol EM^* as executed by a trusted third party, which we later replace by implementing $\mathcal{F}_{\mathsf{EM}^*}$ with MPC. We iteratively select subranges of universe U w.r.t. DP median via the exponential mechanism. After $s = \lceil \log_k |U| \rceil$ steps the last selected subrange contains only the DP median. We split ε, also called *privacy budget*, into s parts such that $\varepsilon = \sum_{j=1}^s \varepsilon_j$, and consume ε_j for each subrange selection. (We describe the budget composition in Section 3.4 and provide a heuristic in Section 5.) Overall, $\mathcal{F}_{\mathsf{EM}^*}$ provides ε-differential privacy:

Theorem 1. *$\mathcal{F}_{\mathsf{EM}^*}$, with privacy parameter ε_j in step $j \in \{1, \dots, s\}$, is ε-differentially private for $\varepsilon = \sum_{j=1}^s \varepsilon_j$.*

Proof. $\mathcal{F}_{\mathsf{EM}^*}$ performs s sequential steps, and each step applies the exponential mechanism $\mathsf{EM}_{u_\mu^c}^{\varepsilon_i}$. Since $\mathsf{EM}_{u_\mu^c}^{\varepsilon_i}$ is $(2\varepsilon_i \Delta u_\mu^c)$-DP [52], with sensitivity $\Delta u_\mu^c = 1/2$ [48], we have ε_i-DP per step. Thus, according to the composition theorem [29], the total privacy budget after all steps is $\sum_{j=1}^s \varepsilon_j$. □

3.4 Accuracy of Differentially Private Median

We express *accuracy* as the absolute error between differentially private and actual median. In more detail, the absolute error is bounded by α with probability at least $1 - \beta$, known as (α, β)-accuracy. Next, we discuss how the data distribution influences accuracy and present worst-case bounds on the accuracy of the exponential mechanism for median selection.

3.4.1 Data Distribution

Accuracy depends on the data distribution, specifically, on *gaps* $d_{i+1} - d_i$, and *duplicates* $d_i = d_j$ with $i \neq j$[5]. Recall, a DP mechanism bounds the probability that data set D and its neighbor D' can be distinguished from the mechanism output. As neighbor D' may contain values from the gaps of D, these gap values must be output with a non-zero probability. This is why bounds for absolute error depend on such gaps between data elements in this and related work (Appendix F). As a worst-case example, consider a data set with universe $U = \{0, 1, \dots, 10^9\}$ containing only an equal number of duplicates for 0 and 10^9. Then, smooth sensitivity is extremely large with 10^9 and the exponential mechanism outputs a value at uniform random. However, for such pathological, worst-case data even the actual median does not provide much insight. On the other hand, the number of duplicates in the data can increase accuracy dramatically. For example, consider a data set where the median has $2c$ duplicates: $d_{n/2 \pm i} = d_{n/2}$ for $i \in \{1, \dots, c\}$. Then, the probability that the exponential mechanism outputs the median is $\exp(c\varepsilon)$ times higher than for any other element. Such duplicates also fit the intuition that the median is a "typical" value from the data that represents it well. In general, the probability to output a "bad" element x decreases exponentially in $\sum c_i$, where $c_i \geq 1$ are duplicate counts of "good" elements y_i, which are closer to the median than x.

[5]To simplify the explanation, assume the universe consists of consecutive integers, i.e., $U = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$ with $a, b \in \mathbb{Z}$.

3.4.2 Accuracy Bounds

In the following, we show that the output of $\mathsf{EM}_u^\varepsilon(D,\mathcal{R})$ contains an element at most $\left\lfloor \frac{\ln(|\mathcal{R}|/\beta)}{\varepsilon} \right\rfloor$ positions away from the median in the sorted data. Note that $|\mathcal{R}|$ is k if we select among k subranges or $|U|$ if we output elements directly.

For our accuracy proofs we structure the universe as a tree: we set U as the root of a tree of height $\log_b |U|$, for some base b, with k child nodes per parent. The child nodes are equal-sized subranges of the parent node and R_i^j denotes the i^{th} subrange in level j.

Theorem 2 (Median Accuracy for Ranges)**.** *Fixing a database D of size n with a set of k subranges $\mathcal{R} = \{R_1^j,\dots,R_k^j\}$ of data universe U. Then, output of $\mathsf{EM}_u^\varepsilon(D,\mathcal{R})$ contains an element at most $\left\lfloor \frac{\ln(k/\beta)}{\varepsilon} \right\rfloor$ positions away from median position $\frac{n}{2}$with probability at least $1-\beta$.*

Our proof uses Corollary 3.12 from [29], which we restate as the following Lemma:

Lemma 1 (Accuracy of the Exponential Mechanism)**.** *Fixing a database D, and let $OPT = \max_{r\in\mathcal{R}} u(D,r)$ denote the maximum utility score of any element $r \in \mathcal{R}$, we have*

$$Pr\left[u(D,\mathsf{EM}_u^\varepsilon(D,\mathcal{R})) \le OPT - \frac{2\Delta u}{\varepsilon}(\ln|\mathcal{R}|+t)\right] \le \exp(-t).$$

Proof of Theorem 2. First, we bound the utility difference between optimal and selected output. Then, we translate this to a bound on the output's rank.

The complementary of Lemma 1 with $\Delta u = \frac{1}{2}$ is

$$\Pr\left[\mathrm{OPT} - u(D,\mathsf{EM}_u^\varepsilon(D,\mathcal{R})) < \frac{\ln|\mathcal{R}|+t}{\varepsilon}\right] > 1-\exp(-t).$$

Let $R_i^j = [r_l, r_u)$ be the output of $\mathsf{EM}_u^\varepsilon(D,\mathcal{R})$. Recall, that for median utility $\mathrm{OPT} = 0$, then,

$$\begin{aligned}\mathrm{OPT} - u(D,\mathsf{EM}_u^\varepsilon(D,\mathcal{R})) &= 0 - u(D,R_i^j)\\ &= \min_{\mathrm{rank}_D(r_l)\le j\le \mathrm{rank}_D(r_u)} \left|j - \frac{n}{2}\right|.\end{aligned}$$

Next, we consider different cases for R_i^j to bound the rank difference between the selected range and the range that contains the median. Assume median $\mu \notin R_i^j$, as otherwise the bound holds trivially, and let d denote the utility difference $\mathrm{OPT} - u(D,\mathsf{EM}_u^\varepsilon(D,\mathcal{R}))$.

For $r_u < \mu$ we have $d = |\mathrm{rank}_D(r_u) - \frac{n}{2}| = \frac{n}{2} - \mathrm{rank}_D(r_u)$ from which we obtain $\mathrm{rank}_D(r_u) > \frac{n}{2} - \frac{\ln|\mathcal{R}|+t}{\varepsilon}$ with probability at least $1-\exp(-t)$. Analog, for $r_l > \mu$ we have $d = \mathrm{rank}_D(r_l) - \frac{n}{2}$, and obtain $\mathrm{rank}_D(r_l) < \frac{n}{2} + \frac{\ln|\mathcal{R}|+t}{\varepsilon}$ with the same probability. Altogether, R_i^j is at most $\left\lfloor \frac{\ln|\mathcal{R}|+t}{\varepsilon} \right\rfloor$ rank positions away from median rank $n/2$ with probability at least $1-\exp(-t)$. We have $k = |\mathcal{R}|$ and setting $\beta = \exp(-t)$ concludes the proof. □

To obtain an absolute error with regards to data elements, consider universe elements instead of subranges as the output of the exponential mechanism.

Corollary 1 (Median Accuracy)**.** *Fixing a sorted database D of size n, let μ be the median of D, and $\widehat{\mu}$ the output of $\mathsf{EM}_u^\varepsilon(D,U)$. Then, absolute error $|\mu - \widehat{\mu}|$ is at most*

$$\max_{i\in\{+1,-1\}\cdot\left\lfloor\frac{\ln(|U|/\beta)}{\varepsilon}\right\rfloor} \left|d_{\frac{n}{2}+i} - d_{\frac{n}{2}}\right|$$

with probability at least $1-\beta$.

The proof follows directly from Theorem 2 with $|\mathcal{R}| = |U|$.

Note that it is more likely to select a "good" subrange as it is to directly select a "good" element from the entire universe (as $k \ll |U|$). However, sequential (subrange) selections consumes ε_j per selection step j which adds up to a total privacy budget of $\varepsilon = \sum_j \varepsilon_j$ as described in Section 3.3. We now show how to choose ε_j to select the subrange containing the median in each iteration step with probability at least $1-\beta$.

Theorem 3 (Choice of ε)**.** *Let $\mathcal{R} = \{R_1^j,\dots,R_k^j\}$, where $R_i^j = [r_l, r_u)$ contains the median, and $n_{ij} = \min\{|\mathrm{rank}_D(\mu) - \mathrm{rank}_D(r_l)|, |\mathrm{rank}_D(r_u+1) - \mathrm{rank}_D(\mu+1)|\}$ is the minimum count of data elements in R_i^j smaller resp. larger than the median. Then, $\mathsf{EM}_u^\varepsilon(D,\mathcal{R})$ selects R_i^j with probability at least $1-\beta$ if*

$$\varepsilon_j \ge \frac{\ln(k/\beta)}{n_{ij}}.$$

Proof. Ranges R_h^j without the median have a rank at least n_{ij} positions away from median rank. More formally,

$$\mathrm{OPT} - u(D,R_h^j) \ge \left|\left(\frac{n}{2} \pm n_{ij}\right) - \frac{n}{2}\right| = n_{ij}.$$

According to Lemma 1 we have $\Pr\left[n_{ij} \ge \frac{\ln|\mathcal{R}|+t}{\varepsilon_j}\right] \le \exp(-t)$. Thus, for $\varepsilon_j \ge \frac{\ln|\mathcal{R}|+t}{n_{ij}}$ the probability that any range R_h^j is selected is at most $\exp(-t)$. We have $k = |\mathcal{R}|$ and setting $\beta = \exp(-t)$ concludes the proof. □

Parameter ε_j is undefined for $n_{ij} = 0$, i.e., when the median is a range endpoint[6]. Note that the exact value of n_{ij} is data-dependent. E.g., for the uniform distribution $n_{ij} \approx |D|/k^j$. A differentially private n_{ij} can be efficiently computed by distributed sum protocols [26, 38, 60, 65] as it is just a count of data elements. However, a differentially private count also consumes a portion of the privacy parameter. For low epsilon (e.g., $\varepsilon = 0.1$) we want to use the entire privacy budget on the actual median selection to achieve high accuracy. Thus, we use a heuristic in our evaluation: larger subranges, that hold exponentially more elements, receive exponentially smaller portions ε_j of the privacy budget (see Section 5 for details).

[6]An undefined ε_j can be avoided by using an additional discretization of the universe, with different subrange endpoints, and switching to it if a (differentially private) check suggests $n_{ij} = 0$ [27].

MPC protocol	*Output / Functionality*
$\mathsf{Rec}(\langle a\rangle)$	a, reconstructed from $\langle a\rangle$
$\mathsf{Add}(\langle a\rangle,\langle b\rangle)$	$\langle a+b\rangle$
$\mathsf{Sub}(\langle a\rangle,\langle b\rangle)$	$\langle a-b\rangle$
$\mathsf{Mul}(\langle a\rangle,\langle b\rangle)$	$\langle a\cdot b\rangle$
$\mathsf{Mod2m}(\langle a\rangle,b)$	$\langle a \mod 2^b\rangle$, where b is public
$\mathsf{Trunc}(\langle a\rangle,b)$	$\langle\lfloor a/2^b\rfloor\rangle$, where b is public
$\mathsf{Rand}(b)$	$\langle r\rangle$ with uniform random b-bit value r
$\mathsf{Choose}(\langle a\rangle,\langle b\rangle,\langle c\rangle)$	$\langle a\rangle$ if bit $c=1$ otherwise $\langle b\rangle$
$\mathsf{LT}(\langle a\rangle,\langle b\rangle)$	$\langle 1\rangle$ if $a<b$ else $\langle 0\rangle$
$\mathsf{Int2FL}(\langle a\rangle)$	converts integer a to secret shared float

Table 2: Basic MPC protocols [5,6] used in EM*. We prefix protocols for integers with Int and floats with FL.

4 MPC for Differentially Private Median

In the following, we describe details of our protocol EM*, which implements ideal functionality $\mathcal{F}_{\mathsf{EM}^*}$, analyse its running time and security.

On a high-level, our protocol recursively selects the best subrange until the DP median is found: First, each party locally *evaluates* a utility score (or weight) for each subrange. They *combine* their results into a global result. Then, they *select* a subrange based on the combined result. We use upper case letters to denote arrays in our protocol, and $A[j]$ denotes the j^{th} element in array A. Our protocol uses integers as well as floating point numbers. We adopt the notation from Aliasgari et al. [5] and represent a floating-point number f as $(1-2s)(1-z)\cdot v\cdot 2^x$ with sign bit s set when the value is negative, zero bit z only set when the value is zero, l_v-bit significand v, and l_x-bit exponent x. The sharing of a floating point value f is a 4-tuple $(\langle v\rangle,\langle x\rangle,\langle s\rangle,\langle z\rangle)$, which we abbreviate as $\langle f\rangle_{\mathsf{FL}}$. To refer to, e.g., the significand v of f we will write $f.v$. (Privacy violations and mitigations w.r.t. limited machine precision are discussed in Appendix D.) The basic MPC protocols used in our protocol are listed in Table 2. We prefix MPC protocols for integers with Int and floating point versions with FL.

4.1 Subrange Selection

On a high level, protocol EM*, implemented in Algorithm 1, computes selection weights for possible outputs (via Algorithm 2) and selects an output according to these weights (via Algorithm 3 or 4). We assume that the universe U and combined data size n are known to all parties (note that the latter can be hidden via padding [3]). Recall, that efficient weight computation and selection are the main challenges for our secure exponential mechanism. Straightforward selection over all universe elements is linear in the size of U. To achieve a running time sublinear in the size of U we selects subranges instead: Algorithm 1 selects one of k subranges based on their median utility. The selected subrange is recursively divided into k subranges until the last subrange, after at most $\lceil\log_k|U|\rceil$ iterations, contains only one element: the differentially private median[7]. Alternatively, one can use fewer selection steps s and select an element from the last subrange at uniform random (line 15 in Algorithm 1). We discuss the running time vs. accuracy trade-offs of reduced selection steps in Section 5. We implement selection with *inverse transform sampling* (ITS) via binary search in Algorithm 2 similar to [30]. ITS uses the uniform distribution to realize any distribution based on its cummulative distribution function. Formally, one draws $r\in(0,1]$ at uniform random and outputs the first $R_j\in\mathcal{R}$ with $\sum_{i=1}^{j-1}\Pr[\mathsf{EM}_u^{\varepsilon}(D,\mathcal{R})=R_i]\le r<\sum_{i=1}^{j}\Pr[\mathsf{EM}_u^{\varepsilon}(D,\mathcal{R})=R_i]$. Recall, we compute unnormalized probabilities (weights), which do not require division for normalization, thus, reducing computation complexity. To use weights instead of probabilities in ITS we only need to multiply r with normalization $N=\sum_{o\in\mathcal{R}}\exp(u(D,o)\varepsilon)$.

Algorithm 1 Algorithm EM*.

Input: Number of subranges k, size n of combined data D, number of selection steps $s\in[1,\lceil\log_k|U|\rceil]$, and $(\varepsilon_1,\dots,\varepsilon_s)$. Data universe U is known to all parties.
Output: Differentially private median of D.

```
1:  $r_l, r_u \leftarrow 0, |U|$
2:  for $j \leftarrow 1$ to $s$ do
3:      $r_\# \leftarrow \max\{1, \lfloor \frac{r_u - r_l}{k} \rfloor\}$
4:      $k \leftarrow \min\{k, r_u - r_l\}$
5:      Define array $W$ of size $k$
6:      if $\varepsilon_j = \ln(2)/2^d$ for some integer $d$ then
7:          $\langle W\rangle_{\mathsf{FL}} \leftarrow \mathsf{Weights}^{\ln(2)/2^d}(r_l, r_u, r_\#, k, n, d)$  //Alg. 3
8:      else
9:          $\langle W\rangle_{\mathsf{FL}} \leftarrow \mathsf{Weights}^*(r_l, r_u, r_\#, k, n, \varepsilon_j)$  //Algorithm 4
10:     end if
11:     $i \leftarrow \mathsf{Select}(\langle W\rangle_{\mathsf{FL}})$  //Algorithm 2
12:     $r_l \leftarrow r_l + (i-1)\cdot r_\#$
13:     $r_u \leftarrow r_l + r_\#$ if $i < k$
14: end for
15: return Uniform random element in $[U[r_l], U[r_u])$
```

We use decomposable utility functions to combine local evaluations over each party's data into a global utility score for the joint data. Next, we present three solutions to efficiently compute weights for decomposable utility functions.

4.2 $\mathsf{Weights}^{\ln(2)}$

We implement $\mathsf{Weights}^{\ln(2)}$ as a special case of our approach $\mathsf{Weights}^{\ln(2)/2^d}$ in Algorithm 3 (with $d=0$ in line 16). Here, parties locally compute *ranks* which are combined into global utility scores. Weights for these scores use a fixed ε of $\ln(2)$ to let us compute 2^u instead of $\exp(\varepsilon\cdot u)$. Solutions for secure exponentiation of 2^u exist where u is an integer or a float

[7]To simplify presentation, assume that $\log_k|U|$ is an integer. Otherwise the last subrange might contain less than k elements, and fewer weight computations are needed in the last step.

Algorithm 2 Algorithm Select.

```
Input: List $\langle W\rangle_{\mathsf{FL}}$ of weights with size $k$.
Output: Index $j \in [1,k]$ sampled according to $\langle W\rangle_{\mathsf{FL}}$.
 1: Define array $M$ of size $k$  //Probability mass
 2: $\langle M[1]\rangle_{\mathsf{FL}} \leftarrow \langle W[1]\rangle_{\mathsf{FL}}$
 3: for $j \leftarrow 2$ to $k$ do
 4:     $\langle M[j]\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLAdd}(\langle W[j]\rangle_{\mathsf{FL}}, \langle M[j-1]\rangle_{\mathsf{FL}})$
 5: end for
 6: $\langle t\rangle \leftarrow \mathsf{IntRand}(b)$  //Bitlength $b$
 7: $\langle f\rangle_{\mathsf{FL}} \leftarrow \mathsf{Int2FL}(\langle t\rangle)$
 8: $\langle x\rangle \leftarrow \mathsf{IntSub}(\langle f.x\rangle, \langle b\rangle)$
 9: $\langle f\rangle_{\mathsf{FL}} \leftarrow (\langle f.v\rangle, \langle x\rangle, \langle f.z\rangle, \langle f.s\rangle)$
10: $\langle r\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle M[k]\rangle_{\mathsf{FL}}, \langle f\rangle_{\mathsf{FL}})$
11: $i_l \leftarrow 1; i_u \leftarrow k$
12: while $i_l < i_u$ do
13:     $i_m \leftarrow \left\lfloor \frac{i_l+i_u}{2} \right\rfloor$
14:     $\langle c\rangle \leftarrow \mathsf{FLLT}(\langle M[i_m]\rangle_{\mathsf{FL}}, \langle r\rangle_{\mathsf{FL}})$
15:     $c \leftarrow \mathsf{Rec}(\langle c\rangle)$
16:     $i_l \leftarrow i_m + 1$ if $c = 1$ else $i_u \leftarrow i_m$
17: end while
18: return $i_l$
```

[5, 7, 20, 43]. When u is an integer (resp. a float) the result 2^u is an integer (resp. float) as well. The complexity of the integer-based solution is linear in the bit-length of u, however, this is not sufficient for us: Recall, that the utility is based on ranks, i.e., counts of data elements, thus u can be roughly as large as the size of the data. An integer representation of 2^u has bit-length u, which is potentially unbounded. Eigner et al. [30] use the float-based solution from [5] but we present a more efficient computation in the following. Although our exponent u is an integer, we do not require the result to be an integer as well. We use the representation of floating point numbers as a 4-tuple to construct a new float to represent 2^u as $(2, u, 0, 0)$, where sign and zero bit are unset, as 2^u cannot be negative or zero. Note that we require no interaction as each party can construct such a float with their share of u. Also, a naive approach requires $2k$ total inputs per party (one per endpoint per k ranges). However, with half-open ranges $[r_l^i, r_u^i)$ in each step i, they overlap for $i > 1$: $r_u^{i-1} = r_l^i$. Thus, the parties only input $k+1$ ranks (Algorithm 3 lines 5, 7).

4.3 Weights$^{\ln(2)/2^d}$

Next, we generalize the weight computation to support $\varepsilon = \ln(2)/2^d$ for integers $d \geq 1$. To illustrate our approach, we implement Weights$^{\ln(2)/2^d}$ in Algorithm 3 for $d = 1$, and describe the approach for any integer d: Recall, our goal is to compute the weight $\exp(\varepsilon u)$ with efficient MPC protocols. As we can efficiently compute $2^{\varepsilon u}$ if εu is an integer, we approximate the weight by truncating εu to an integer before exponentiation with base 2. To avoid a loss of precision we correct this approximation with a multiplicative term based on the truncated remainder. More formally, with ε as above

Algorithm 3 Algorithm Weights$^{\ln(2)/2^d}$.

```
Input: Range $[r_l, r_u)$, subrange size $r_\#$, number $k$ of subranges, data
    size $n$, and parameter $d \in \{0,1\}$. Subrange ranks $\mathrm{rank}_{D_p}(\cdot)$ are
    input by each party $p \in \{1,\ldots,m\}$.
Output: List of weights.
 1: Define arrays $R$ of size $k+1$, $W$ of size $k$; initialize $R$ with zeros
 2: for $p \leftarrow 1$ to $m$ do  //Get input from each party
 3:     for $j \leftarrow 1$ to $k$ do  //Divide range into $k$ subranges
 4:         $i_l \leftarrow r_l + (j-1) \cdot r_\#$
 5:         $\langle R[j]\rangle \leftarrow \mathsf{IntAdd}(\langle R[j]\rangle, \langle \mathrm{rank}_{D_p}(U[i_l])\rangle)$
 6:     end for
 7:     $\langle R[k+1]\rangle \leftarrow \mathsf{IntAdd}(\langle R[k+1]\rangle, \langle \mathrm{rank}_{D_p}(U[r_u])\rangle)$
 8: end for
 9: for $j \leftarrow 1$ to $k$ do
10:     $\langle u_u\rangle \leftarrow \mathsf{IntSub}(\langle R[j+1]\rangle, \langle \frac{n}{2}\rangle)$
11:     $\langle u_l\rangle \leftarrow \mathsf{IntSub}(\langle \frac{n}{2}\rangle, \langle R[j]\rangle)$
12:     $\langle c_u\rangle \leftarrow \mathsf{IntLT}(\langle R[j+1]\rangle, \langle \frac{n}{2}\rangle)$
13:     $\langle c_l\rangle \leftarrow \mathsf{IntLT}(\langle \frac{n}{2}\rangle, \langle R[j]\rangle)$
14:     $\langle t\rangle \leftarrow \mathsf{IntChoose}(\langle u_u\rangle, \langle 0\rangle, \langle c_u\rangle)$
15:     $\langle u\rangle \leftarrow \mathsf{IntChoose}(\langle u_l\rangle, \langle t\rangle, \langle c_l\rangle)$
16:     if $d = 0$ then
17:         $\langle W[j]\rangle_{\mathsf{FL}} \leftarrow (\langle 2\rangle, \langle u\rangle, \langle 0\rangle, \langle 0\rangle)$  //float $\langle 2^u\rangle$
18:     else
19:         $\langle t\rangle \leftarrow \mathsf{IntTrunc}(\langle u\rangle, d)$
20:         $\langle e\rangle_{\mathsf{FL}} \leftarrow (\langle 2\rangle, \langle t\rangle, \langle 0\rangle, \langle 0\rangle)$
21:         $\langle c\rangle \leftarrow \mathsf{IntMod2m}(\langle u\rangle, d)$
22:         $\langle s\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLChoose}(\langle 1\rangle_{\mathsf{FL}}, \langle \sqrt{2}\rangle_{\mathsf{FL}}, \langle c\rangle)$
23:         $\langle W[j]\rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle e\rangle_{\mathsf{FL}}, \langle s\rangle_{\mathsf{FL}})$
24:     end if
25: end for
26: return $\langle W\rangle_{\mathsf{FL}}$
```

the weight for u is

$$2^{u/2^d} = 2^{\lfloor u/2^d \rfloor} \cdot 2^{(u \mod 2^d)/2^d}.$$

First, we compute $2^{\lfloor u/2^d \rfloor}$ (lines 19–21 in Algorithm 4). Then, we multiply this with one of 2^d constants of the form $2^{(u \mod 2^d)/2^d}$. E.g., for $d = 1$, we either use 1, if u is even, or $\sqrt{2}$ otherwise (line 22). The constants themselves are not secret and can be pre-computed. Which constant was selected, leaks the last d bits from u, thus, we choose them securely.

4.4 Weights*

We implement Weights* in Algorithm 4. To allow arbitrary values for ε we avoid costly secure exponentiation for weight computation altogether: Utility u, decomposable w.r.t. u', allows for efficient combination of local weights for D_i, input by the parties, into global weights for D via multiplication (as described in Section 3.2).

4.5 Running Time Complexity Analysis

We analyse the running time of EM* w.r.t. MPC protocols from Table 2 (omitting non-interactive addition/subtraction),

Algorithm 4 Algorithm Weights*.

```
Input: Range $[r_l, r_u)$, subrange size $r_\#$, number $k$ of subranges, data
    size $n$, and $\varepsilon$. Subrange weights $e^{\varepsilon(\cdot)}$ are input by each party
    $p \in \{1,\ldots,m\}$.
Output: List of weights.
 1: Define arrays $W^l$, $W^u$, $W$ of size $k$; initialize $W^l$, $W^u$ with ones
 2: for $p \leftarrow 1$ to $m$ do  //Get input from each party
 3:   for $j \leftarrow 1$ to $k$ do  //Divide range into $k$ subranges
 4:     $i_l \leftarrow r_l + (j-1) \cdot r_\#$
 5:     $i_u \leftarrow r_u$ if $j = k$ else $r_l + j \cdot r_\#$
 6:     $\langle W^l[j] \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle W^l[j] \rangle_{\mathsf{FL}}, \langle e^{\varepsilon\left(\frac{|D_p|}{2} - \mathrm{rank}_{D_p}(U[i_l])\right)} \rangle_{\mathsf{FL}})$
 7:     $\langle W^u[j] \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLMul}(\langle W^u[j] \rangle_{\mathsf{FL}}, \langle e^{\varepsilon\left(\mathrm{rank}_{D_p}(U[i_u]) - \frac{|D_p|}{2}|\right)} \rangle_{\mathsf{FL}})$
 8:   end for
 9: end for
10: for $j \leftarrow 1$ to $k$ do
11:   $\langle c_u \rangle \leftarrow \mathsf{FLLT}(\langle W^u[j] \rangle_{\mathsf{FL}}, \langle 1 \rangle_{\mathsf{FL}})$
12:   $\langle c_l \rangle \leftarrow \mathsf{FLLT}(\langle W^l[j] \rangle_{\mathsf{FL}}, \langle 1 \rangle_{\mathsf{FL}})$
13:   $\langle t \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLChoose}(\langle W^u[j] \rangle_{\mathsf{FL}}, \langle 1 \rangle_{\mathsf{FL}}, \langle c_u \rangle)$
14:   $\langle W[j] \rangle_{\mathsf{FL}} \leftarrow \mathsf{FLChoose}(\langle W^l[j] \rangle_{\mathsf{FL}}, \langle t \rangle_{\mathsf{FL}}, \langle c_l \rangle)$
15: end for
16: return $\langle W \rangle_{\mathsf{FL}}$
```

and their complexity is given in Appendix C. We measure the running time of our implementation in Section 5.

Theorem 4. EM* *with* Weights$^{\ln(2)}$ *or* Weights$^{\ln(2)/2^d}$ *requires $O(k\lceil \log_k |U| \rceil)$ MPC protocol calls, with* Weights* *we require $O(mk\lceil \log_k |U| \rceil)$. Note that complexity of these MPC protocols is at most $O(l_v \log l_v + l_x)$ for bit-lengths l_v, l_x (Appendix C).*

Proof. EM* invokes the weight computation and Select at most $\lceil \log_k |U| \rceil$ times. An invocation of Weights$^{\ln(2)}$ or Weights$^{\ln(2)/2^d}$ performs k truncations IntTrunc, $2k$ comparisons IntLT and $2k$ selections IntChoose. Additionally, Weights$^{\ln(2)/2^d}$ also requires one truncation IntTrunc, modulo IntMod2m, float selection FLChoose and float multiplication FLMul. Weight computation via Weights* requires $2km$ float multiplications FLMul, $2k$ float comparisons FLLT and $2k$ float selections FLChoose. Each invocation of Select requires $k-1$ float additions FLAdd, only one random draw IntRand, conversion Int2FL and float multiplication FLMul. Also, Select performs at most $\log_2(k)$ comparisons FLLT and share reconstruction steps during binary search. □

4.6 Security

We consider the *semi-honest model* introduced by Goldreich [36] where corrupted protocol participants do not deviate from the protocol but gather everything created during the run of the protocol. Our protocol consists of multiple subroutines realized with MPC protocols listed in Table 2 (for details and security proof references we refer to [6]). To analyze the security of the entire protocol we rely on the well-known *composition theorem* [36, Section 7.3.1]. Basically, MPC protocols using an ideal functionality (a subroutine provided by a trusted third party) remain secure if the ideal functionality is replaced with an MPC protocol implementing the same functionality. We implement such ideal functionality with the maliciously secure SCALE-MAMBA framework [6] (which was faster than its semi-honest fork in a WAN, as detailed in Appendix E). Our protocol performs multiple subrange selections and each selection round is maliciously secure. Overall, we only provide semi-honest security as malicious adversaries can deviate from inputs provided in previous rounds. We later show how to extend our protocol to malicious adversaries, but first we proof semi-honest security for EM*:

Theorem 5. *Protocol* EM* *realizes $\mathcal{F}_{\mathsf{EM}^*}$ in the presence of semi-honest adversaries.*

Proof. To prove semi-honest security we show the existence of a *simulator* Sim according to Goldreich [36] such that the distributions of the protocol transcript EM* is computationally indistinguishable from simulated transcript using $\mathcal{F}_{\mathsf{EM}^*}$ produced in an "ideal world" with a trusted third party. Note that an adversary in the ideal world learns nothing except the protocol inputs and outputs, hence, if he cannot distinguish simulated transcripts (from ideal world) and actual transcripts (in the real world), he learns nothing in our real-world implementation. Next, we formalize the ideal and real-world executions, ideal and real, with notation from Evans et al. [32]: Consider a subset of corrupted parties $C \subset \mathcal{P}$, and let $\mathtt{VIEW}_i$ denote the view of party $i \in C$ during the execution of EM* implementing ideal functionality $\mathcal{F}_{\mathsf{EM}^*}$, including all exchanged messages and internal state, and let x_i denote the protocol input of party P_i and $\widehat{\mu}$ the final output of all parties. The parameters s, k, U are public. Then, $\mathsf{real}_{\mathsf{EM}^*}$, on input security parameter κ, C and all x_i, runs protocol EM* (where each party P_i behaves honestly using its own input x_i) and outputs $\{\mathtt{VIEW}_i | i \in C\}, \widehat{\mu}$. And $\mathsf{ideal}_{\mathcal{F}_{\mathsf{EM}^*},\mathsf{Sim}}$, with the same inputs, computes $\widehat{\mu} \leftarrow \mathcal{F}_{\mathsf{EM}^*}(x_1,\ldots,x_m)$ and outputs $\mathsf{Sim}(C, \widehat{\mu}, \{x_i \mid i \in C\}), \widehat{\mu}$. Now, simulator Sim produces a transcript for $\mathsf{real}_{\mathsf{EM}^*}$ as follows: As we operate on secret shares, which look random to the parties [32], Sim replaces all secret shares with random values to create $\mathtt{VIEW}_i$. Likewise, the secret-shared output of the weight computations (Algorithm 3 and 4) are replaced with randomness. Sim can simulate Algorithm 2 by recursively splitting U into k subranges, and outputting the subrange containing $\widehat{\mu}$ in each selection step. Finally, Sim outputs a uniform random element from the last subrange (Algorithm 1). Altogether, a semi-honest adversary cannot learn more than the (ideal-world) simulator as this information is sufficient to produce a transcript of our (real-world) protocol. □

For malicious adversaries, we need to ensure consistency between rounds based on Aggarwal et al. [3], who securely compute the (non-DP) median via comparison-based pruning rounds. Informally, we have two consistency constraints:

First, valid rank inputs must be monotone within a step. Second, for consistency between steps, valid inputs are contained in the subrange output in the previous step. Formally, let $\{R_1^i, \dots, R_k^i\}$ denote the set of subranges in the i^{th} step of EM^* and let l_j^i, u_j^i denote the lower resp. upper range endpoint of R_j^i. Then, $\text{rank}_{D_p}(l_1^i) \leq \text{rank}_{D_p}(l_2^i) \leq \dots \leq \text{rank}_{D_p}(l_k^i) \leq \text{rank}_{D_p}(u_k^i)$ describes monotone input in step i for party p. Consistency between step i and $i+1$, if the j^{th} range was selected, is expressed as $\text{rank}_{D_p}(l_1^{i+1}) = \text{rank}_{D_p}(l_j^i)$ and $\text{rank}_{D_p}(u_k^{i+1}) = \text{rank}_{D_p}(u_j^i)$. In other words, the subrange output in the previous step is used in the current step. Analogously, we can enforce consistency for weights as they are based on rank values.

4.7 Scaling to Many Parties

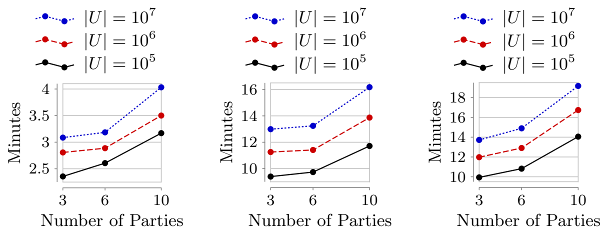
Recall, we distinguish two sets of parties: *Input parties* send shares of their input to *computation parties* which run the secure computation on their behalf. The latter can be a subset of the input parties or non-colluding untrusted servers (e.g., multiple cloud service providers). This scales nicely as the number of computation parties is independent of the number of input parties and can be constant, e.g., 3. In our evaluation (Section 5) $m \in \{3,6,10\}$ computation parties perform the computation for 10^6 input parties, each holding a single datum. Addition suffices for $\mathsf{Weights}^{\ln(2)}$ and $\mathsf{Weights}^{\ln(2)/2^d}$ to combine local rank values into a global rank. Addition is essentially "free" as it requires no interaction between the computation parties. For $\mathsf{Weights}^*$ we require multiplication to combine the local weights, which requires interaction during the preprocessing step. However, $\log n$ rounds suffice to combine the inputs by building a tree of pairwise multiplications with 2^i multiplications at level i [5].

5 Evaluation

Our implementation is realized with the SCALE-MAMBA framework [6] using Shamir secret sharing with a 128-bit modulus and honest majority. Next, we evaluate the running time, privacy budget and accuracy of our solution and refer to Appendix E for additional evaluations.

5.1 Running Times

We performed our evaluation on t2.medium AWS instances with 2GB RAM, 4 vCPUs [8] and the Open Payments data set from the Centers for Medicare & Medicaid Services (CMS) [33]. Our evaluation uses 10^6 records from the Open Payments data set, however, our approach scales to any data set size as we consider universe subranges. We used the maximum number of selection steps, i.e., $s = \lceil \log_k |U| \rceil$, with $k = 10$ ranges per step. We evaluated the average running time of 20 runs of the *entire* protocol EM^*, i.e., offline as well as online phase (see Appendix E), in a LAN and a WAN.



(a) $\mathsf{Weights}^{\ln(2)}$ (b) $\mathsf{Weights}^{\ln(2)/2^d}$, $d=2$ (c) $\mathsf{Weights}^*$

Figure 4: Average running time of EM^* – with weight computation subroutines $\mathsf{Weights}^{\ln(2)}$, $\mathsf{Weights}^{\ln(2)/2^d}$, or $\mathsf{Weights}^*$– for 20 runs on t2.medium instances in Ohio and Frankfurt (100 ms delay, 100 Mbits/s bandwidth).

LAN: We measured our running time for 3 parties in a LAN with 1 Gbits/s bandwidth to compare it to Eigner et al. [30] who only report LAN running times. We support universe sizes of more than 5 orders of magnitude larger with comparable running times: They compute weights per elements and require around 42 seconds for $|U| = 5$, whereas our protocol EM^* using $\mathsf{Weights}^{\ln(2)}$/ $\mathsf{Weights}^{\ln(2)/2^d}$/ $\mathsf{Weights}^*$ runs in approx. 11 / 33 / 64 seconds for $|U| = 10^5$. For detailed measurements see Table 4 in Appendix E.

WAN: We consider m computation parties, which already received and combined secret-shared inputs from 10^6 users (Section 4.7), and report the average running time of our protocol. We split the m parties into two regions, Ohio (us-east-2) and Frankfurt (eu-central-1), and measured an inter-region round time trip (RTT) of approx. 100 ms with 100 Mbits/s bandwidth. We evaluated all weight computation subroutines in Figure 4 for $m \in \{3,6,10\}$ computation parties and $|U| \in \{10^5, 10^6, 10^7\}$. The results are very stable, as the 95% confidence intervals deviate by less than 0.5% on average. $\mathsf{Weights}^{\ln(2)}$ (Figure 4a) is the fastest with running times around 3 minutes for 3 parties, whereas $\mathsf{Weights}^{\ln(2)/2^d}$ (Figure 4b) and $\mathsf{Weights}^*$ (Figure 4c) require around 13 and 14 minutes respectively. However, we consider large universe sizes (billions of elements) in a real-world network with large latency. The choice of weight computation enables a trade-off between faster running times, i.e., $\mathsf{Weights}^{\ln(2)}$ with fixed ε, and smaller privacy loss ε, i.e, $\mathsf{Weights}^*$, with $\mathsf{Weights}^{\ln(2)/2^d}$ positioned in the middle (faster running time than $\mathsf{Weights}^*$ with smaller ε compared to $\mathsf{Weights}^{\ln(2)}$). The number k of subranges allow a similar trade-off, as discussed next.

5.2 Privacy Budget vs. Running Time

The *privacy budget* is the sum of privacy parameters consumed per step, i.e., the overall privacy loss. Figure 5 shows

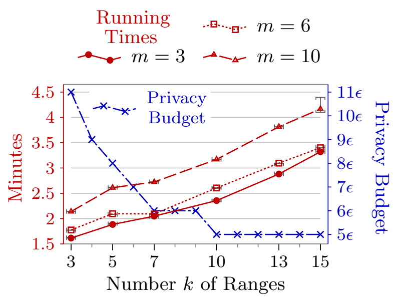


Figure 5: Privacy vs. running time trade-off: For increasing number k of subranges the running time (left axis) increases whereas the consumed privacy budget (right axis) decreases. (Illustrated for EM* with Weights$^{\ln(2)}$ and $|U| = 10^5$).

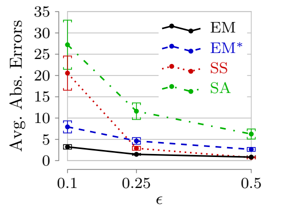
how the privacy budget and the running time are affected by the number k of subranges. Larger k leads to larger running times, as the number of costly secure computations depends on the number of ranges times the number of selection steps ($k \cdot \lceil \log_k |U| \rceil$), which increases proportionally to k. However, smaller values for k require more selection steps ($\lceil \log_k |U| \rceil$), which lead to an increase in the privacy budget. Overall, for $k = 10$ subranges, as used in our evaluation, the consumed privacy budget is small with an acceptable running time.

5.3 Accuracy Comparison to Related Work

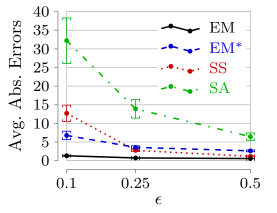
EM* performs multiple selection steps s, each consume a portion ε_i of the overall privacy budget $\varepsilon = \sum_{i=1}^{s} \varepsilon_i$. How to optimally split ε (optimal composition) is #P-complete [55]. Thus, we use the following heuristic to divide ε among the selection steps: Initial steps cover exponentially larger subranges, and require exponentially less of the privacy budget. After a while an equal split is more advantageous, as the subranges become smaller and contain fewer elements. Altogether, we use $\varepsilon_i = \varepsilon/2^{s-i+1}$ if $i \leq \lfloor s/2 \rfloor$ and $\varepsilon_i = \varepsilon'/(s - \lfloor s/2 \rfloor)$ else, where ε' is the remaining privacy budget. We used $s = \lceil \log_k |U| \rceil - 1$ for our accuracy evaluation. We found in our experiments that performing one selection step less increases accuracy, as the privacy budget can be better divided among the other remaining steps and the last subrange is already small enough (at most k elements).

Related work computing DP median in the central model shows a strong data dependence which makes straightforward comparison difficult (Appendix F). Therefore, we empirically evaluated the different approaches closest to ours, i.e., supporting more than 2 parties, on real-world data sets [42, 64, 67] as well as the normal distribution in Figure 6[8] for 100 averaged runs with 95%-confidence intervals. Low ε (as evaluated) is desirable as it provides more privacy or allows the remaining privacy budget to be spend on additional queries. The

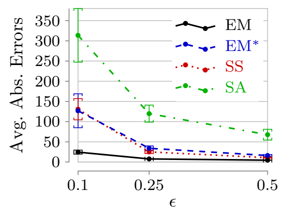
[8]"Small" data is the most challenging regime for DP [15, 56], thus, we use small data sets to better illustrate the accuracy differences.



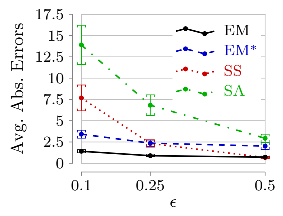
(a) Credit card data [67], first 10^5 payment records in Cents.



(b) Walmart supply chain data [42], 175k shipment weights as integers.



(c) California public salaries [64], 71k records, state department's total wages.



(d) Normal distribution with $\sigma = 3$, 10^5 samples (as integers with scaling factor 1000).

Figure 6: Comparing exponential mechanism (EM) as baseline, this work (EM*), smooth sensitivity (SS) [58], sample-and-aggregate (SA) [59] on different data, 100 averaged runs.

evaluation for smooth sensitivity [58] and exponential mechanism per element assume a trusted party with full access to the data set, whereas our approach and [59] use MPC instead of a trusted party. Nissim et al. [58] (SS in Figure 6) compute instance-specific additive noise, requiring full data access, and achieve good accuracy, however, the exponential mechanism can provide better accuracy for low ε. Pettai & Laud [59] (SA in Figure 6) securely compute the noisy average of the 100 values closest to the median within a clipping range. Recall, the median is the 0.5^{th}-percentile. To minimize the error from clipping range $[c_l, c_u]$, we choose $c_l = 0.49^{\text{th}}$-percentile, $c_u = 0.51^{\text{th}}$-percentile, i.e., we presume to already know a tight range for the actual median. Nonetheless, in our experiments the absolute error of SA is the largest. Overall, no solution is optimal for all ε and data sets. However, the exponential mechanism EM, and our protocol EM*, provide the best accuracy for low ε, i.e., high privacy, compared to additive noise approaches [58, 59].

6 Related Work

Next, we describe related work for secure computation of the exponential mechanism, DP median and decomposability.

Secure Exponential Mechanism: Alhadidi et al. [4] present a secure 2-party protocol for the exponential mechanism for max utility functions. It uses garbled circuits and oblivious polynomial evaluation to compute Taylor series for the exponential function. Our work is more general as we support more parties and a broader class of utility functions, including max utility functions. Eigner et al. [30] present a carefully designed secure exponential mechanism in the multi-party setting. Their work is more general, supporting arbitrary utility functions and malicious parties, but they are linear in the size of the universe, and securely compute the exponential function. We provide a sublinear solution without costly secure exponentiation, supporting at least 5 orders of magnitude more elements than them. Böhler and Kerschbaum [14] also securely compute the DP median with the exponential mechanism. They optimize their protocol for the 2-party setting, compute the utility over (sorted) data, and provide DP for small data (sublinear in the size of the universe). They initially prune large data sets via [3] (who securely compute the exact median), requiring a relaxation of DP [39], to achieve running time sublinear in the universe size. We consider the multi-party setting and provide pure differential privacy.

DP Median: Pettai and Laud [59] securely compute DP statistics, including the DP median, via sample-and-aggregate [58]. Their implementation is based on secret sharing in a 3-party setting. Pettai and Laud [59] compute the DP median as noisy average of 100 values closest to the median within a clipping range, which limits accuracy, especially, if the data contains outliers or large gaps (see Section 5.3). Dwork and Lei [27] consider robust privacy-preserving statistics with a trusted third party where data samples are known to be drawn i.i.d. from a distribution. They present the first DP median algorithm that does not require bounds for the data but aborts if the data are not from a "nice" distribution with small sensitivity. Their DP median algorithm first estimates scale s via DP interquartile range and the noise magnitude $sn^{-1/3}$ can be large. Nissim et al. [58] present smooth sensitivity, which analyzes the data to provide instance-specific noise. For the DP median, the exponential mechanism provides better accuracy for low epsilon and can be efficiently computed, whereas computation of smooth sensitivity requires full data access in clear or the error increases (see Section 2.1.2).

PINQ, a DP query framework developed by McSherry [51], also computes the DP median via the exponential mechanism, however, they rely on a trusted third party with access to the data in clear. Cryptε [19] employs two non-colluding untrusted servers and cryptographic primitives to compute noisy histograms (Laplace mechanism) for SQL queries (e.g., count, distinct count) in the central model, which can be extended to compute the median. However, we show that the exponential mechanism is more accurate for the median with low ε. Also, Cryptε has a running time linear in the data size, whereas our work is independent of the data size. Smith et al. [63] and Gaboardi et al. [34] consider the restrictive non-interactive local model, where at most one message is sent from client to server, and achieve optimal local model error. However, local DP requires more samples to achieve the same accuracy as central DP. (No non-interactive LDP-protocol [34, 63] can achieve asymptotically better sample complexity than $O(\varepsilon^{-2}\alpha^{-2})$ for error α [24].) We, on the other hand, are interested in high accuracy, as in the central model, even for small sample sizes. We give accuracy bounds for related work for the DP median in the central model in Appendix F. As these data-dependent bounds are hard to compare we provide an empirical comparison in Section 5.3.

Decomposability: MapReduce is a programming paradigm for distributed data aggregation where a mapper produces intermediary results (e.g., partial sums) that a reducer combines into a result (e.g., total sum). Airavat [61] provide a Hadoop-based MapReduce programming platform for DP statistics based on additive noise (Laplace mechanism) with an untrusted mapper but trusted reducer. We consider decomposable utility functions for the exponential mechanism without any trusted parties. The secure exponential mechanisms [4, 30] use decomposable utility functions (max and counts), but do not classify nor provide optimizations for such functions. Blocki et al. [16] minimize cummulative error for DP password frequency lists employing (decomposability of) frequencies for their dynamic programming, which has access to all the data in the clear. We use decomposable aggregate functions to efficiently and securely combine inputs.

7 Conclusion

We presented a novel alternative for differentially private median computation with high accuracy (even for small number of users), without a trusted party, that is efficiently computable (practical running time) and scaleable (sublinear in the size of the universe). Our semi-honest multi-party protocol implements the exponential mechanism for decomposable aggregate functions (e.g., rank-based statistics) as used in MapRedue-style algorithms, and can be extended to malicious parties. For the median, the exponential mechanism provides the best utility vs. privacy trade-off for low ε in our evaluations of related work in the central model.

We optimize our protocol for decomposable functions (allowing efficient MPC on distributed data), and use efficient alternatives to exponentiations for floating-point numbers. We implemented our protocol in the SCALE-MAMBA framework [6], and evaluated it for 1 million users using 3 semi-honest computation parties achieving a running time of seconds in a LAN, and 3 minutes in a WAN (100 ms latency, 100 Mbits/s bandwidth).

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333 (MOSAICrOWN).

References

[1] WWDC 2016. Engineering privacy for your users, 2016.

[2] John M. Abowd. The u.s. census bureau adopts differential privacy. In *Proceedings of the annual ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD, 2018.

[3] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the median (and other elements of specified ranks). *Journal of Cryptology*, 2010.

[4] Dima Alhadidi, Noman Mohammed, Benjamin CM Fung, and Mourad Debbabi. Secure distributed framework for achieving ε-differential privacy. In *International Symposium on Privacy Enhancing Technologies Symposium*, PETS, 2012.

[5] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. NDSS, 2013.

[6] Abdelrahaman Aly, Marcel Keller, Dragos Rotaru, Peter Scholl, Nigel P. Smart, and Tim Wood. Scale–mamba documentation. https://homes.esat.kuleuven.be/~nsmart/SCALE/, 2020.

[7] Abdelrahaman Aly and Nigel P Smart. Benchmarking privacy preserving scientific operations. In *International Conference on Applied Cryptography and Network Security*, ACNS, 2019.

[8] Amazon.com. Amazon Web Services. https://aws.amazon.com/ec2/pricing/on-demand/.

[9] Victor Balcer and Albert Cheu. Separating local & shuffled differential privacy via histograms, 2019.

[10] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2014.

[11] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, CRYPTO, 1991.

[12] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2011.

[13] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

[14] Jonas Böhler and Florian Kerschbaum. Secure sublinear time differentially private median computation. In *Network and Distributed Systems Security Symposium*, NDSS, 2020.

[15] Andrea Bittau, Ulfar 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 Symposium on Operating Systems Principles*, SOSP, 2017.

[16] Jeremiah Blocki, Anupam Datta, and Joseph Bonneau. Differentially private password frequency lists. In *Network and Distributed Systems Security Symposium*, NDSS, 2016.

[17] Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, SCN, 2010.

[18] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2019.

[19] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the annual ACM SIGMOD International Conference on Management of data*, SIGMOD, 2020.

[20] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, TCC, 2006.

[21] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual International Cryptology Conference*, CRYPTO, 2012.

[22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[23] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, NIPS, 2017.

[24] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2013.

[25] Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages, and Programming*, ICALP, 2006.

[26] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2006.

[27] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the annual ACM symposium on Theory of Computing*, STOC, 2009.

[28] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, TCC, 2006.

[29] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 2014.

[30] Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially private data aggregation with optimal utility. In *Proceedings of the Annual Computer Security Applications Conference*, ACSAC, 2014.

[31] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the annual ACM conference on computer and communications security*, CCS, 2014.

[32] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2018.

[33] Centers for Medicare & Medicaid Services. Complete 2017 program year open payments dataset, 2017.

[34] Marco Gaboardi, Adam Smith, and Jinhui Xu. Empirical risk minimization in the non-interactive local model of differential privacy.

[35] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. Preserving differential privacy under finite precision semantics. In *Theoretical Computer Science*, TCS, 2016.

[36] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. 2009.

[37] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the annual ACM symposium on Theory of Computing*, STOC, 1987.

[38] Slawomir Goryczka and Li Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE transactions on Dependable and Secure Computing*, 2017.

[39] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the annual ACM conference on Computer and Communications Security*, CCS, 2017.

[40] Justin Hsu, Sanjeev Khanna, and Aaron Roth. Distributed private heavy hitters. In *International Colloquium on Automata, Languages, and Programming*, ICALP, 2012.

[41] Christina Ilvento. Implementing the exponential mechanism with base-2 differential privacy, 2019.

[42] Kaggle.com. Walmart supply chain: Import and shipment. `https://www.kaggle.com/sunilp/walmart-supply-chain-data/data`, 2018. Retrieved: October, 2019.

[43] Liina Kamm. *Privacy-preserving statistical analysis using secure multi-party computation*. PhD thesis, PhD thesis, University of Tartu, 2015.

[44] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 2011.

[45] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. `https://eprint.iacr.org/2020/521`.

[46] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT, 2018.

[47] Marcel Keller, Dragos Rotaru, Nigel P Smart, and Tim Wood. Reducing communication channels in mpc. In *International Conference on Security and Cryptography for Networks*, SCN, 2018.

[48] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. Differential privacy: From theory to practice. *Synthesis Lectures on Information Security, Privacy, & Trust*, 2016.

[49] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 2009.

[50] Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil Vadhan. The limits of two-party differential privacy. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2010.

[51] Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the annual ACM SIGMOD International Conference on Management of data*, SIGMOD, 2009.

[52] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 2007.

[53] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the annual ACM conference on computer and communications security*, CCS, 2012.

[54] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Annual International Cryptology Conference*, CRYPTO, 2009.

[55] Jack Murtagh and Salil Vadhan. The complexity of computing the optimal composition of differential privacy. In *Theory of Cryptography Conference*, TCC, 2016.

[56] Seth Neel, Aaron Roth, Giuseppe Vietri, and Zhiwei Steven Wu. Differentially private objective perturbation: Beyond smoothness and convexity, 2019.

[57] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Annual International Cryptology Conference*, CRYPTO, 2012.

[58] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the annual ACM symposium on Theory of Computing*, STOC, 2007.

[59] Martin Pettai and Peeter Laud. Combining differential privacy and secure multiparty computation. In *Proceedings of the Annual Computer Security Applications Conference*, ACSAC, 2015.

[60] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the annual ACM SIGMOD International Conference on Management of data*, SIGMOD, 2010.

[61] Indrajit Roy, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce.

[62] Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.

[63] Adam Smith, Abhradeep Thakurta, and Jalaj Upadhyay. Is interaction necessary for distributed private learning? In *IEEE Symposium on Security and Privacy*, SP, 2017.

[64] Gaurav Sood. California Public Salaries Data, 2018.

[65] Hassan Takabi, Samir Koppikar, and Saman Taghavi Zargar. Differentially private distributed data analysis. In *IEEE International Conference on Collaboration and Internet Computing*, CIC, 2016.

[66] Apple's Differential Privacy Team. Learning with privacy at scale, 2017.

[67] Machine Learning Group ULB. Credit card fraud detection, 2018.

[68] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Annual IEEE Symposium on Foundations of Computer Science*, FOCS, 1986.

A Distributed Differential Privacy

The original definition of Differential Privacy considers the central model with unbounded adversaries [25,28] (see Definition 1), later work expanded it to a distributed setting [26,44], and considered computationally-bounded parties [54].

We consider multiple computationally-bounded, semi-honest parties performing a joint secure computation realized with (t,m)-secret sharing. The following definition from [30] fits our setting, where $\texttt{VIEW}^p_\Pi(D)$ denotes the view of party p during the execution of protocol Π on input D, including all exchanged messages and internal state, and λ is a security parameter:

Definition 7 (Distributed Differential Privacy)**.** *A randomized protocol Π implemented among m computation parties $\mathcal{P} = \{P_1,\dots,P_m\}$, achieves* distributed differential privacy *w.r.t. a coalition $C \subset \mathcal{P}$ of semi-honest computation parties of size t, if the following condition holds: for any neighbors D,D' and any possible set S of views for protocol Π,*

$$Pr\left[\texttt{VIEW}^C_\Pi(D) \in S\right] \leq \exp(\varepsilon) \cdot Pr\left[\texttt{VIEW}^C_\Pi(D') \in S\right] + negl(\lambda).$$

The definition can be expanded to a malicious setting by replacing the semi-honest parties $\mathcal{P}$ and semi-honestly secure protocol Π with malicious parties and a maliciously secure protocol. Note that the negligible function $\text{negl}(\lambda)$ can be omitted for protocols providing information-theoretic security (such as standard secret sharing), however, our implementation in SCALE-MAMBA provides computational security (due to the online phase as described in Section 2.2).

Protocol	*Rounds*	*Interactive Operations*
Rec	1	1
IntRand	0	0
IntMod2m	$O(1)$	$O(t)$
IntTrunc	4	$4t+1$
IntLT	4	$4b-2$
Int2FL	$\log v+13$	$\log v(2v-3)-11$
FLAdd	$O(\log v)$	$O(v\log v+x)$
FLMul	$8v+10$	11
FLLT	6	$4v+5x+4\log x+13$

Table 3: Complexity of MPC protocols for b-bit integers, t-bit truncation modulus, and floats with v-bit significand and x-bit exponent [5, 6, 17, 30].

B Equality of Definitions 5 and 6

We show equality of Definitions 5 and 6 with proof by cases. Consider range $R=[r_l,r_u)$ and its position relative to the median μ for Definition 5: Case *i)* For $r_u<\mu$ we have $\text{rank}_D(r_u)<n/2$, thus, $u_\mu=-|\text{rank}_D(r_u)-n/2|=\text{rank}_D(r_u)-n/2$. Case *ii)* For $r_l>\mu$ we have $\text{rank}_D(r_l)>n/2$, thus, $u_\mu=n/2-\text{rank}_D(r_l)$. Case *iii)* Otherwise, the range contains the median, i.e., $u_\mu=0$.

Note that it suffices to look at r_l in case i) (resp., r_u in case ii)), as $\text{rank}_D(r_l)\leq\text{rank}_D(r_u)$ and the range endpoint closest to μ defines the utility for the range. Definition 6 uses the same cases and is an alternative way to express Definition 5.

C Complexity of MPC Protocols

Table 3 lists the complexities for MPC protocols typically measured in the number of *rounds* and *interactive operations*, where rounds describes the count of sequential interactive operations, and interactive operations (e.g., reconstruct sharing, multiplications) require each party to send messages to all other parties. We omit integer addition/subtraction as these operations are non-interactive and the parties can perform them locally. Share reconstruction is denoted with Rec. Note that Choose$(\langle a\rangle,\langle b\rangle,\langle c\rangle)$ is implemented with one multiplication and two additions $(b+(a-b)\cdot c)$, and that IntRand uses correlated randomness already exchanged in the offline phase (hence zero interaction and rounds).

D Precision and Privacy

In general, DP mechanisms operate on reals, whereas actual implementations are limited to the precision of physical machines. However, limited precision can lead to privacy violations: As shown by Mironov [53], the Laplace mechanism is vulnerable to finite precision as the set of possible outcomes can differ between neighboring databases due to irregularities caused by floating point implementations. Their proposed mitigation is to perform "snapping", roughly, they clamp the noisy result to a fixed range and ensure that they output is evenly spaced. Recent work by Ilvento [41] extend this line of study to the exponential mechanism, which is also vulnerable to finite precision. The suggested mitigation is switching from base e to base 2 to perform precise arithmetic by, e.g., for integer utility functions, approximating ε as $\eta=-\log_2(x/2^y)$ for integers x,y such that $x/2^y\leq 1$.

Interestingly, their mitigation is similar to our efficient secure computation. Our implementation is based on an integer utility function and Weights$^{\ln(2)}$ uses base 2 for efficiency reasons and is not vulnerable to such attacks. We can strengthen Weights$^{\ln(2)/2^d}$, with $\varepsilon=\ln(2)/2^d$, by using randomized rounding for non-interger utilities as described in [41, Section 3.2.2] if we omit $1/2^d$ from ε and include it as a factor in the utility definition (making the utility non-integers). For Weights*, which supports arbitrary ε, careful choices for ε as described above mitigate attacks on limited precision.

Implementation Note: SCALA-MAMBA's floating point numbers (`sfloat`) are associated with a statistical security parameter κ satisfying $\kappa<b-2\cdot l_v$ where b is the bit-length of the modulus and l_v is the bit-length of the significand. Security with $\kappa=40$ is the default for $b=128$ and we use $l_v=40$ in our evaluation, to support large utility values.

Protocol	$\|U\|$	*Running time*
Eigner et al. [30]	5	42.3 s
EM* & Weights$^{\ln(2)}$	10^5	11.3 s (7.7 s)
	10^6	13.5 s (9.2 s)
	10^7	15.4 s (10.7 s)
EM* & Weights$^{\ln(2)/2^d}$, $d=2$	10^5	33.7 s (23.6 s)
	10^6	39.8 s (27.8 s)
	10^7	46.8 s (31.4 s)
EM* & Weights*	10^5	64.3 s (41.6 s)
	10^6	77.3 s (52.4 s)
	10^7	91.8 s (61.1 s)

Table 4: Running times for 3 parties in a 1 Gbits/s LAN for this work and Eigner et al. [30]. We report the average of 20 runs on t2.medium instances with 4 vCPUs, 2 GB RAM (and r4.2xlarge instances with 8 vCPUs, 61 GB RAM). Eigner et al. [30] evaluated on a 3.20 GHz, 16 GB RAM machine.

E Additional Evaluation

The online and offline phase are integrated in newer versions of SCALE-MAMBA, thus, we only provide measurements for the total protocol, i.e., offline as well as online phase.

Running time in a LAN: We compare our running time to

Protocol	$\lvert U\rvert$	*Communication* $m=3$	$m=6$	$m=10$
EM* & Weights$^{\ln(2)}$	10^5	178 MB	402 MB	1.41 GB
	10^6	202 MB	448 MB	1.54 GB
	10^7	222 MB	497 MB	1.75 GB
EM* & Weights$^{\ln(2)/2^d}$, d=2	10^5	634 MB	1.38 GB	4.73 GB
	10^6	748 MB	1.63 GB	5.58 GB
	10^7	866 MB	1.88 GB	6.39 GB
EM* & Weights*	10^5	664 MB	1.56 GB	5.59 GB
	10^6	785 MB	1.83 GB	6.57 GB
	10^7	907 MB	2.11 GB	7.59 GB

Table 5: Communication cost (WAN with 100 Mbits/s and 100 ms latency): Data sent per party, average of 20 runs for $m \in \{3,6,10\}$ parties and $|U| \in \{10^5, 10^6, 10^7\}$.

Work	*Error bound* α *with* $Pr[\mathit{abs.\ error} \le \alpha] \ge 1-\beta$
Nissim et al. [58]	$\max\limits_{k=0,..,n} e^{-k\varepsilon} \max\limits_{t=0,..,k+1} \left(d_{\frac{n}{2}+t} - d_{\frac{n}{2}+t-k-1}\right)\gamma$
Dwork and Lei [27]	$\frac{d_{\lceil 0.75n\rceil} - d_{\lceil 0.25n\rceil}}{n^{1/3}}\gamma$
Pettai and Laud [59]	$\left(\frac{c_u - c_l}{j} + \frac{\max(U)-\min(U)}{\varepsilon\exp(\Omega(\varepsilon\sqrt{j}))}\right)\gamma$
This work	$\max_{i\in\{+1,-1\}\cdot\left\lfloor\frac{\ln(\lvert U\rvert/\beta)}{\varepsilon}\right\rfloor} \left\lvert d_{\frac{n}{2}+i} - d_{\frac{n}{2}}\right\rvert$

Table 6: DP median methods in the central model with $\gamma = \ln(1/\beta)/\varepsilon$. Data $D \in U^n$ is sorted and the error terms are simplified to ease comparison: assuming expected sensitivity [27], shortened approximation error term [59] (see [58, Th. 4.2]), using one selection step for this work. Here, $[c_l, c_u]$ is the presumed clipping range for the j elements closest to the median [59].

Eigner et al. [30], which only report running times in a LAN, in Table 4. Eigner et al. [30] evaluated their protocol with a sum utility function on a 3.20 GHz, 16 GB RAM machine. They are linear in the size of the universe and compute weights for a very small universe of only 5 elements. We, on the other hand, are sublinear in the size of the universe as we compute weights per subrange and use efficient alternatives to costly secure exponentiation. We evaluated universe sizes at least 5 order of magnitudes larger than [30] with comparable running times: Our running time for Weights$^{\ln(2)}$, Weights$^{\ln(2)/2^d}$ is below [30] on rather modest t2.medium instances (4 vCPUs, 2 GB RAM) for universe size $|U| = 10^5$. Even if we also consider weights per element (i.e., subrange size 1) for any decomposable utility function our protocols compute at least 6 times more weights per second on t2.medium instances. (E.g., for $k = 10, |U| = 10^5$ and Weights* we compute 50 weights in 64.3 seconds, i.e., 0.78 weights per second, compared to 0.12 for [30].)

We also evaluated our protocol on r4.2xlarge instances (8 vCPUs, 61 GB RAM), see seconds in parenthesis in Table 4. In a LAN the running time compared to t2.medium instances is reduced by at least 30%, however, in a WAN setting the latency plays a more important role than powerful hardware and the running times are much closer. Thus, we only present running times for t2.medium instances in a WAN in Section 5.

Communication: The communication for the maximum number of steps ($\lceil \log_{10}|U| \rceil$) in a WAN (100 Mbits/s with 100 ms latency) is detailed in Table 5. For 3 parties and one billion universe elements, the communication for Weights$^{\ln(2)}$ is 222 MB per party, for Weights$^{\ln(2)/2^d}$ it is 866 MB, and Weights* requires 907 MB. We stress that this communication is required for *malicious security* in each round as provided by the SCALE-MAMBA implementation. MP-SPDZ [45], a fork of SCALE-MAMBA's predecessor SPDZ2, also provides semi-honest security. MP-SPDZ with semi-honest security requires much less communication, e.g., only around 25 MB for 3 parties, $|U| = 10^5$, and Weights*. However, the running time in a WAN was some minutes slower compared to SCALE-MAMBA in our tests (presumably due to SCALE-MAMBA's batched communication rounds and integrated online and offline phases, where parallel threads create offline data "just-in-time" [6, 7]). Thus, regarding our protocol, one can choose efficiency w.r.t. communication (MP-SPDZ) or running time (SCALE-MAMBA).

Cost of Malicious Security: To achieve malicious security, via consistency checks as detailed in Section 4.6, we require additional running time and communication. For the maximum number of steps with one billion universe elements in a WAN (100 Mbits/s with 100 ms latency) EM* with Weights* additionally needs around 10/10/12 minutes and 0.65/1.4/5 GB for 3/6/10 parties. EM* with Weights$^{\ln(2)}$ or Weights$^{\ln(2)/2^d}$ ($d = 2$) additionally requires around 1.3/1.5/2 minutes and 115/260/825 MB for 3/6/10 parties.

F Accuracy Bounds: Related Work for Multiparty DP Median

We list theoretical accuracy bounds for related work, i.e., computation of the DP median in the central model supporting many parties, in Table 6. Note that the table entries, except for this work, are the sensitivity of the method multiplied by factor γ[9] (and with an additional error term for [59]). Hence, the first entry is the definition of smooth sensitivity for the median (multiplied by γ). For an empirical comparison of this work with related work we refer to Section 5.3.

[9]The related works draw additive noise r from zero-centered Laplace distribution with scale s/ε and sensitivity s (Laplace mechanism, Definition 2). Since $\Pr[|r| < t \cdot s/\varepsilon] = 1 - \exp(-t)$ [29, Fact 3.7], we can bound the absolute error $|r|$ as in Table 6 by setting $\beta = \exp(-t), \gamma = t/\varepsilon = \ln(1/\beta)/\varepsilon$.