Secure Sublinear Time Differentially Private Median Computation

1st Jonas Böhler
SAP Security Research
Karlsruhe, Germany
jonas.boehler@sap.com

2nd Florian Kerschbaum
University of Waterloo
Waterloo, Canada
florian.kerschbaum@uwaterloo.ca

***Abstract*—In distributed private learning, e.g., data analysis, machine learning, and enterprise benchmarking, it is commonplace for two parties with confidential data sets to compute statistics over their combined data. The median is an important robust statistical method used in enterprise benchmarking, e.g., companies compare typical employee salaries, insurance companies use median life expectancy to adjust insurance premiums, banks compare credit scores of their customers, and financial regulators estimate risks based on loan exposures.**

The *exact* median can be computed securely, however, it leaks information about the private data. To protect the data sets, we securely compute a *differentially private* median over the joint data set via the exponential mechanism. The exponential mechanism has a runtime linear in the data universe size and efficiently sampling it is non-trivial. 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 utility as the central model, where noise is only applied once by a trusted server.

We present an efficient secure computation of a differentially private median of the union of two large, confidential data sets. Our protocol has a runtime sublinear in the size of the data universe and utility like the central model without a trusted third party. We provide differential privacy for small data sets (sublinear in the size of the data universe) and prune large data sets with a relaxed notion of differential privacy providing limited group privacy. We use dynamic programming with a static, i.e., data-independent, access pattern, achieving low complexity of the secure computation circuit. We provide a comprehensive evaluation over multiple AWS regions (from Ohio to N. Virgina, Canada and Frankfurt) with a large real-world data set with a practical runtime of less than 7 seconds for millions of records.

I. Introduction

In distributed private learning two parties A, B, with confidential data sets D_A, D_B respectively, want to compute statistics of their combined data. Example applications are data analysis, machine learning, collaborative forecasting and enterprise benchmarking. The median is an important *robust* statistical method, i.e., a few outliers in the data do not skew the result. The median is used to represent a "typical" value from a data set and is utilized in enterprise benchmarking, where companies measure their performance against the competition to find opportunities for improvement. Businesses compare, e.g., typical employee salaries per department, bonus payments or sales incentives to better assess their attractiveness for the labor market, and insurance companies use the median life expectancy to adjust insurance premiums. Further, banks compare credit scores of their customers, and financial regulators estimate risks based on loan exposures.

Since the data are sensitive, e.g., salary or health information, the parties want to compute the median without revealing any of their data to each other. A solution to reveal the exact median and nothing else was presented by Aggarwal et al. [1], however, the exact median itself is a value from either D_A or D_B, and, as shown in [13, 45], median queries can be used to uncover the exact value of targeted individuals. To protect the data sets and hinder targeted inference attacks we also use *differential privacy* [16, 20]. Inference attacks [13, 45] rely on median values from the actual data set. The differentially private median, however, is a non-deterministic value from the entire data universe and yet it is close to the actual median with high probability. For small data sets (sublinear in the size of the data universe) we provide differential privacy, and for large data sets we first prune the input using the relaxed notion of differential privacy introduced in [28]. Instead of considering *neighbors*, i.e., data sets differing in one record, the relaxed notion requires neighbors to also have the same output w.r.t. the initial input pruning. However, we provide empirical evidence that the relaxation is not too restrictive on real-world data sets [11, 33, 51, 54]. A trusted third party, called *curator* in differential privacy literature [18], can implement any differentially private algorithms. However, this trusted party requires full access to the unprotected data. To protect the inputs without relying on a trusted third party we use *secure computation* [24], i.e., the parties run a protocol to compute a function on their respective inputs such that nothing about their input is revealed except the function result. Google reported using secure computation to link online ads with offline purchases [7, 32], and government institutes use it to detect tax fraud [8] and perform studies (e.g., [9]). In our case, we securely compute the differentially private median via the exponential mechanism, as it provides the best accuracy vs. privacy trade-off for low ϵ (see our discussion in Section II). The exponential mechanism from McSherry and Talwar [39] selects a specific value, like the median, from a data universe $\mathcal{U}$, has a computation complexity linear in the size of the entire data universe [39] and efficiently sampling it is non-trivial [18]. Also, the exponential mechanism requires exponentiations and

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24150
www.ndss-symposium.org

divisions, increasing the secure computation complexity. Pettai and Laud [44] securely compute the differentially private median using the framework by Nissim et al. [43]. Unlike their work we also considered network delay and used modest hardware[1] and our protocol is still 13 times faster for millions of records with a latency of 25 ms.

We present an efficient protocol to securely compute the differentially private median of the union of two large, confidential data sets with computation complexity sublinear in the size of the data universe. First, the parties prune their own data in a way that maintains their median. Then, they sort and merge the pruned data. The sorted data is used to compute selection probabilities for the entire data universe. Finally, the probabilities are used to select the differentially private median. To optimize the runtime of our protocol we use dynamic programming for the probability computation with a static, i.e., data-independent, access pattern, achieving low complexity of the secure computation circuit. We utilize different cryptographic techniques, garbled circuits as well as secret sharing, to combine their respective advantages, namely, comparisons and arithmetic computations. We simplify the probability and sampling computations to minimize direct access to the data, which reduces secure computation overhead. Furthermore, we compute the required exponentiations for the exponential mechanism without any secure computation.

In summary, the contributions of our protocol combining secure computation and differential privacy are

- selection of the differentially private median of the union of two distributed data sets without revealing anything else about the data,
- an improved runtime complexity sublinear in the size of the data universe achieved by data-independent dynamic programming and input pruning for large data sets,
- a comprehensive evaluation with a large real-world data set with a practical runtime of less than 7 seconds for millions of records even with 100 ms network delay and 100 MBits/s bandwidth.

We note that our protocol can be easily adapted to securely compute the differentially private p^{th}-percentile, i.e., the value larger than $p\%$ of the data. The remainder of this paper is organized as follows: In Section II we detail the problem description. In Section III we describe preliminaries for our dynamic programming protocol. In Section IV we explain our approach and introduce definitions. Then, we present our protocol and implementation details for the secure computation of the differentially private median in Section V. We provide a detailed performance evaluation in Section VI. We describe related work in Section VII and conclude in Section VIII.

II. Problem Description

We consider the problem of two parties computing the differentially private median over their combined data sets. Next, we describe implementation models and basic techniques for differentially private algorithms.

[1]Our evaluation is performed with AWS t2.medium instances (4 vCPUs, 2GB RAM) compared to the 12-core 3GHZ CPU, 48GB RAM setup of [44].

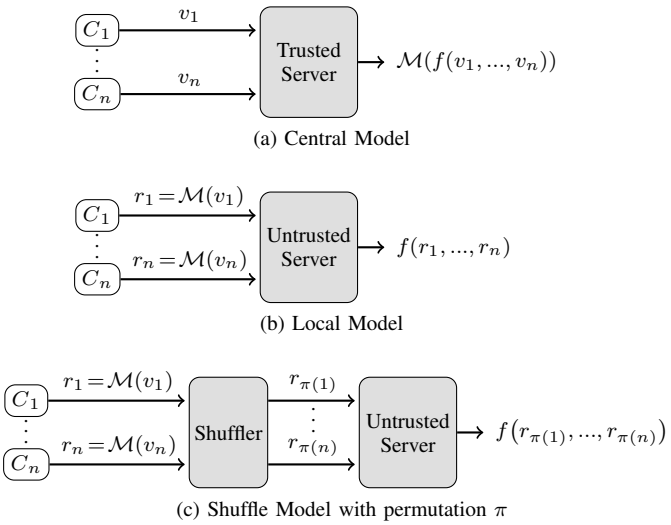
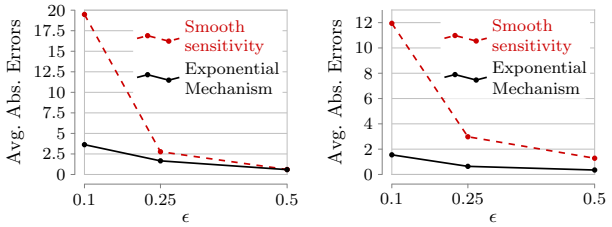


Fig. 1. Models for differentially private algorithms $\mathcal{M}$. Client C_i sends a message – raw value v_i or randomized r_i – to a server. The server computes some function f over the messages, and releases the differentially private result.

A. Models for Differentially Private Algorithms

Differentially private algorithms $\mathcal{M}$ can be implemented in different models which are visualized in Figure 1. 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 differentially private algorithms, is only applied once. In the *local model* (Figure 1b), introduced by [34], clients apply $\mathcal{M}$ locally and send anonymized values to an untrusted server for aggregation. The accuracy is limited as multiple randomizations occur. It requires enormous amounts of data, compared to the central model, to achieve acceptable accuracy bounds [5, 12, 30, 38]. Specifically, an exponential separation between local and central model for accuracy and sample complexity was shown by [34]. Recently, an intermediate *shuffle model* (Figure 1c) was introduced [5, 12]: An additional party is added between clients 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 the 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 [12]. As our goal is high accuracy without additional parties, this work dismisses the shuffle model.

To combine the benefits of the local and central model, namely, high accuracy and strong privacy, *secure computation* [24] is used in related work [19, 25, 48, 52]. Secure computation allows to simulate central model algorithms in the local model. Secure computation is a cryptographic protocol run between the clients which only reveals the computation's output and nothing more about their sensitive data. Hence, secure computation of the median is superior to distributed computation methods that reveal additional statistics (e.g., histograms or prefix query results) from which to compute a (noisy) median. As Smith et al. [50] note, general techniques that combine secure computation and differential privacy suffer



(a) Credit Card data [54], first 10^5 payment records in Cents.

(b) Walmart Supply Chain data [33], 175k shipment weights as integers.

Fig. 2. Absolute errors, averaged for 100 differentially private median computations via exponential mechanism and Laplace mechanism with smooth sensitivity for $\epsilon \in \{0.1, 0.25, 0.5\}$.

from bandwidth and liveness constraints which render them impractical for large data sets. Our contribution is an optimized secure protocol for the differentially private median that runs in seconds on million of records in real-world networks.

B. Differential Privacy Techniques

Informally, the main techniques to provide differential privacy are *additive noise*, e.g., the Laplace mechanism [18], and *probabilistic selection*, namely, the exponential mechanism [39]. To compute the differentially private median we use the exponential mechanism [39], which provides selection probabilities for possible median values. A simpler, but less accurate, alternative is the *Laplace mechanism* [18], which adds noise, sampled from the Laplace distribution, to a function result, i.e., $f(D) + \text{Laplace}\,(\Delta f/\epsilon)$. The noise depends on Δf, the *sensitivity* of the function, and a privacy parameter ϵ formalized later. 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. [43], additionally analyzes the data to provide instance-specific additive noise which is often much smaller. Li et al. [35] note that the Laplace mechanism is ineffective for the median as the sensitivity, and thus noise, can be high. As mentioned before, the accuracy in the local model is limited [5, 30, 38]. However, even in the central model with smooth sensitivity the exponential mechanism is usually more accurate. To demonstrate this we evaluated the absolute error of the Laplace mechanism with smooth sensitivity and the exponential mechanism for real-world data sets [33, 54] in Fig. 2. In general, large differences between elements close to the median or small ϵ, which corresponds to strong privacy guarantees, increase noise magnitudes and thus errors even with smooth sensitivity. Furthermore, secure computation of smooth sensitivity requires access to the entire dataset or the error further increases[2], which prohibits sublinear secure computation with high accuracy. Thus, our reason for using the exponential mechanism to compute the median is two-fold: It provides the best (known) accuracy for small ϵ, and, as we will show, it can be implemented as sublinear-time secure computation.

[2]Smooth sensitivity approximations exist that provide a factor of 2 approximation in linear-time, or an additive error of $\max(\mathcal{U})/\text{poly}(n)$ in sublinear-time [43, Section 3.1.1]. Note that this error e is w.r.t. smooth sensitivity s and the additive noise is even larger with Laplace $((s+e)/\epsilon)$.

III. Preliminaries

Next we introduce preliminaries for differential privacy and secure computation, and some notation.

A. Notation

We model a database as $D = \{d_0, d_1, \ldots, d_{n-1}\} \in \mathcal{U}^n$. We call $\mathcal{U}$ *data universe* and assume it to be an integer range, i.e., $\mathcal{U} = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$ with $a, b \in \mathbb{Z}$. We note that rational numbers can be expressed as integers via *fixed-point number representation*.[3] To simplify the description we assume the size n of D to be even which can be ensured by padding. Then, the median is the value $d_{n/2-1}$ in sorted D. We denote with $I_D = \{0, \ldots, n-1\}$ the set of indices for D and refer to non-distinct data elements as *duplicates*, i.e., $d_i = d_j$ with $i \neq j$ $(i, j \in I_D)$. We apply union under bag semantics, i.e., $D_A \cup D_B$ is a *bag* containing elements from $\mathcal{U}$ as often as they appear in data sets D_A and D_B combined[4]. We treat the difference of two bags, $D_A \backslash D_B$, as a *set* containing only elements from D_A that are not also in D_B.

B. Differential Privacy

Differential privacy introduced by Dwork, McSherry, Nissim, and Smith [16, 20] is a privacy notion, adopted by major technology companies [15, 22, 53, 55]. Differential privacy enables one to learn statistical properties of a data set while protecting the privacy of any individual contained in it. Data sets D, D' are called *neighbors* or *neighboring*, denoted with $D \simeq D'$, when data sets D can be obtained from D' by adding or removing one element, i.e., $D = D' \cup \{x\}$ with $x \in \mathcal{U}$ or $D = D' \backslash \{y\}$ with $y \in D'$.

Informally, a differentially private algorithm limits the impact that the presence or absence of any individual's data in the input database can have on the *distribution* of outputs. The formal definition is as follows:

Definition 1 (Differential Privacy)**.** *A mechanism $\mathcal{M}$ satisfies ϵ-differential privacy, where $\epsilon \geq 0$, if for all neighboring data sets D and D', and all sets $S \subseteq$ Range$(\mathcal{M})$*

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

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 polynomial-time bounded adversary. Mironov et al. [41] define indistinguishable computationally differential privacy (IND-CDP) for two-party computation (2PC) with computationally bounded parties. The presented definition is according to [28] for parties A, B with data sets D_A, D_B, privacy parameters ϵ_A, ϵ_B and security parameter λ. Furthermore, $\texttt{VIEW}_A^\Pi$ denotes the view of A during the execution of protocol Π.

Definition 2 (IND-CDP-2PC)**.** *A two-party protocol Π for computing function f satisfies $(\epsilon_A(\lambda), \epsilon_B(\lambda))$-indistinguishable computationally differential privacy (IND-CDP-2PC) if $\texttt{VIEW}_A^\Pi(D_A, \cdot)$ satisfies $\epsilon_B(\lambda)$-IND-CPA, i.e.,*

[3]A binary number of bit-length b can represent $d \in \mathbb{Q}$ as $d' \in \mathbb{Z}$ if $d = d' \cdot 2^{-f}$ with $-2^{b-1} + 1 \leq d' \leq 2^{b-1} - 1$ and scaling factor 2^{-f}, $f \in \mathbb{N}$.

[4]This interpretation of union is equivalent to the *sum* function for bags.

for any probabilistic polynomial-time (in λ) adversary $\mathcal{A}$, for any neighboring data sets (D_B, D'_B)

$$Pr[\mathcal{A}(\mathtt{VIEW}_A^{\Pi}(D_A, D_B)) = 1]$$
$$\leq \exp(\epsilon_B) \cdot Pr[\mathcal{A}(\mathtt{VIEW}_A^{\Pi}(D_A, D'_B)) = 1] + negl(\lambda).$$

Likewise for B's view for any neighbors (D_A, D'_A) and ϵ_A.

For notational convenience let $\epsilon = \epsilon_A = \epsilon_B$. We operate in the *semi-honest* model [24] (also called honest-but-curious) where participants do not deviate from the protocol but try to extract as much information from the protocol transcript as possible. A protocol is consider secure in the semi-honest model when the transcript does not reveal anything beyond the computed functionality.

C. f-neighboring

He et al. [28] introduced the notion of *f-neighbors*: neighbors that also have the same output w.r.t. to a function f. For our security proof we require f-neighboring and adapt it to our scenario.

Definition 3 (f-Neighbor)**.** *Given function $f : \mathcal{U}^k \times \mathcal{U}^l \to \mathcal{O}$, $k, l \in \mathbb{N}$, and $D_A \in \mathcal{U}^k$. Data sets D_B and D'_B are f-neighbors w.r.t. $f(D_A, \cdot)$ if*

1) *they are neighbors, and*
2) *$f(D_A, D_B) = f(D_A, D'_B)$.*

f-neighboring for D_B is similarly defined.

In [28] f-neighboring is applied to record matching, where neighbors differ in at most one *non-matching* record. In our scenario f is input pruning, the first step of our protocol which reduces the input set size and we denote it as PRUNE. PRUNE is a partial execution of comparison-based pruning from [1] described in Section IV-D. We distinguish two forms of pruning: deterministic and randomized. *Deterministic* pruning, such as PRUNE, might differ between neighboring data sets and thus potentially violate differentially privacy for its common neighboring notion. By considering PRUNE-neighbors, where pruning outputs are the same, neighboring data sets cannot be distinguished, and differential privacy holds. To verify that PRUNE-neighboring is not too restrictive and can be used in real-world applications we evaluated neighboring data sets from real-world data sets [11, 33, 51, 54] and found they are all also PRUNE-neighboring albeit with limited group privacy. See Section VI for details of the experiment. In *randomized* pruning each comparison result is randomized. The probability that the half of the data containing the median is never discarded decreases exponentially in the number of comparisons [29]. Hence, accuracy is significantly impacted with high probability and we dismiss randomized pruning in favor of PRUNE-neighboring.

D. Exponential Mechanism

The exponential mechanism, introduced by McSherry and Talwar [39], expands the application of differential privacy to functions with non-numerical output, and when the output is not robust to additive noise. The exponential mechanism selects a result from a fixed set of outputs $\mathcal{O}$ while satisfying differential privacy. The mechanism is exponentially more likely to select "good" results where "good" is quantified via a utility function $u(D, o)$ which takes as input a data set $D \in \mathcal{U}^n$ and a potential output $o \in \mathcal{O}$. The utility function provides a utility score for o w.r.t. D and all possible output values from $\mathcal{O}$. Informally, a higher score means the output is more desirable and its selection probability is increased accordingly. The formal definition is according to [35].

Definition 4 (Exponential Mechanism)**.** *For any utility function $u : (\mathcal{U}^n \times \mathcal{O}) \to \mathbb{R}$ and a privacy parameter ϵ, the exponential mechanism $\mathcal{M}_u^{\epsilon}(D)$ outputs $o \in \mathcal{O}$ with probability proportional to $\exp\left(\frac{\epsilon u(D,o)}{2\Delta u}\right)$, where*

$$\Delta u = \max_{\forall o \in \mathcal{O}, D \simeq D'} |u(D, o) - u(D', o)|$$

is the sensitivity of the utility function. That is,

$$Pr[\mathcal{M}_u^{\epsilon}(D) = o] = \frac{\exp\left(\frac{\epsilon u(D,o)}{2\Delta u}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\frac{\epsilon u(D,o')}{2\Delta u}\right)}. \quad (1)$$

Median Utility Function: We focus on the median and use the median utility function from Li et al. where $\text{rank}_D(x)$ denotes the number of elements in D smaller than x.

Definition 5 (Median utility function)**.** *The median utility function $u_{med} : (\mathcal{U}^n \times \mathcal{U}) \to \mathbb{Z}$ gives a utility score for each $x \in \mathcal{U}$ w.r.t. $D \in \mathcal{U}^n$ as*

$$u_{med}(D, x) = -\min_{\text{rank}_D(x) \leq j \leq \text{rank}_D(x+1)} \left| j - \frac{n}{2} \right|.$$

Note that for the median $\mathcal{O} = \mathcal{U}$, i.e., every universe element has to be considered as a potential output. The sensitivity of u_{med} is $1/2$ since adding an element increases $n/2$ by $1/2$ and j either increases by 1 or remains the same [35]. Thus, the denominator $2\Delta u$ in the exponents of Equation (1) equals 1, and we will omit it in the rest of this work. The intuition behind this utility definition is to use the rank of elements to quantify their "closeness" to the median. The median itself has the highest utility value, 0, all other elements have negative utility. The further away an element in a sorted data set (i.e., its rank) is from the median position, the smaller its utility. Note that Definition 5 can be adapted to select elements of arbitrary rank k, e.g., to find the 25^{th}- and 75^{th}-percentile. In this work we focus on the secure computation of the differentially private median but this can easily be extended to securely compute the differentially private k^{th}-ranked element.

E. Secure Computation

We use secret sharing as well as garbled circuits as addition and scalar multiplication are more efficient with the former whereas comparisons can be more efficiently implemented as boolean circuits with the latter.

Additive Secret Sharing: We require all values to be in the ring $\mathbb{Z}_{2^b}$ and perform all operations modulo $\mathbb{Z}_{2^b}$. In additive p-out-of-p secret sharing a party P_i, $1 \leq i \leq p$, (or a separate dealer) "splits" its secret value $s \in \mathbb{Z}_{2^b}$ into p shares and all shares are required to reconstruct the secret. First, P_i creates uniformly random values $s_1, \ldots, s_{p-1} \in \mathbb{Z}_{2^b}$. Then, P_i sets $s_p = s - \sum_{i=1}^{p-1} s_i$. Intuitively, a shared secret is reconstructed by adding all shares together, i.e., $s = \sum_{i=1}^{p} s_i$. Privacy

follows from the fact that shares $s_1, \ldots, s_p$ are uniformly random and the sum of any strict subset of the shares is also random. We denote the sharing of s as $\langle s \rangle = (s_1, \ldots, s_p)$. Addition with shared secret values $\langle s \rangle, \langle r \rangle$ is straightforward since $\langle s \rangle + \langle r \rangle = (s_1 + r_1, \ldots, s_p + r_p)$, as is multiplication with a public value $t \in \mathbb{Z}_{2^b}$ where $t\langle s \rangle = (ts_1, \ldots, ts_p)$. We also write $\langle s \rangle_{P_j}$ instead of s_j to highlight that it is P_j's share of s. In our implementation we use the ring $\mathbb{Z}_{2^{64}}$ as computations modulo 2^{64} are commonly supported on standard CPUs.

Garbled Circuits: A garbled circuit, first described by Yao [56], is a general technique to securely evaluate any function by implementing it as a boolean circuit and "garbling" each gate's truth table. Informally speaking, given two parties, the four possible inputs of a garbled table are not plaintext bits but random labels. One party is the *garbler* who garbles the gates and creates the labels. The other party, called *evaluator*, receives the garbled circuit and evaluates it. The garbler includes all her input labels in the garbled circuit (which look random to the evaluator). However, the garbler cannot learn the evaluator's input and cannot send both input labels per gate to the evaluator, otherwise the garbler's input will be revealed. To solve this problem 1-out-of-2 *oblivious transfer* (OT) [23, 47] is used: The evaluator receives only her input label and the garbler remains oblivious. Given the input labels for both parties the evaluator can determine (decrypt) the output label for a gate and use it as input for the next gate. An output translation table, also provided by the garbler, maps the final random output label to the plain result. For a formalized description see Appendix A.

IV. Building Blocks for DP Median Selection

We implement an efficient, secure computation of the exponential mechanism which selects the differentially private median from the entire data universe $\mathcal{U}$. There are two challenges for secure computation of the exponential mechanism:

- the runtime complexity is linear in $|\mathcal{U}|$ as probabilities for *all* possible outputs in $\mathcal{U}$ are computed,
- the general exponential mechanism is too inefficient with general secure computation as it requires $|\mathcal{U}|$ exponentiations and divisions.

In this section we present building blocks for our practically efficient, sub-linear time protocol overcoming those challenges.

A. Overview

For now we focus on a single data set as we later prune and merge the data sets from the two parties into one data set. For data set D with universe $\mathcal{U}$ we compute the median selection probabilities for all of $\mathcal{U}$ using only D by utilizing dynamic programming. To compute the probabilities efficiently we first define a simplified utility function $utility$, which computes utility for all universe elements but only requires D as input, in Section IV-B. The simplified $utility$ provides incorrect utility scores in the presence of duplicates. Thus, we define gap to discard these incorrect scores and compute the median selection probabilities, denoted as $weight$. The sum of these probabilities is the basis for the cumulative distribution function, which we denote with $mass$. Then, we sample the differentially private median based on $mass$ with

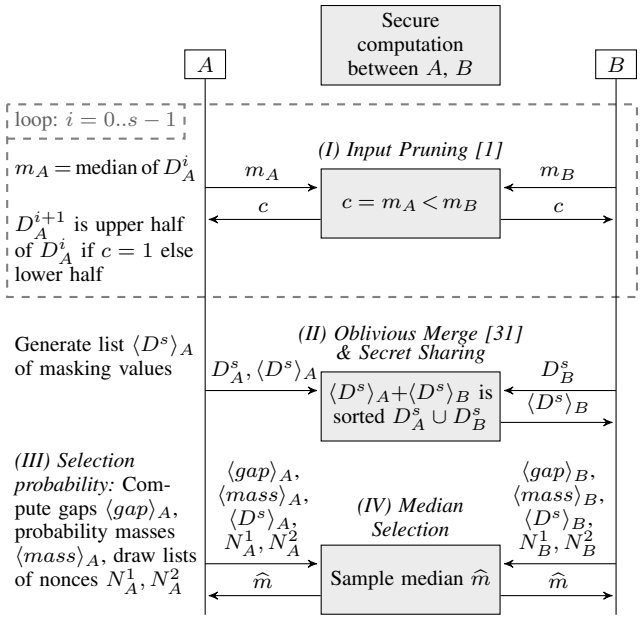


Fig. 3. High-level protocol overview with comments for A, where s is the number of pruning steps, D_A^0 is sorted D_A, and $\langle D^s \rangle_A, \langle gap \rangle_A, \langle mass \rangle_A$ are A's shares for all values d_i^s, gaps $gap(i)$, and masses $mass(i)$ respectively ($i \in \{0, \ldots, |D^s| - 1\}$).

inverse transform sampling described in Section IV-C. To further reduce secure computation complexity we prune the input D in Section IV-D. A high-level overview of our protocol is visualized in Fig. 3, and we present our full protocol in Section V. In the first step, the parties prune their input. Then, they securely merge and secret share their pruned data. In the third step they compute selection probabilities and, in the last step, sample the differentially private median.

Note that in the following we define gap, $utility$, and $weight$ such that direct access to the data D – and therefore the need for secure computation – is minimized: Each party can compute $utility$ and $weight$ without any access to D. Furthermore, gap has a static access pattern in dynamic programming, independent of the elements in (sorted) D, which makes the gap function *data-oblivious*, i.e., an attacker who sees the access pattern cannot learn anything about the sensitive data.

B. Utility with Static Access Pattern

Recall that the exponential mechanism evaluates the utility function u_{med} for *all* elements in the data universe $\mathcal{U}$. However, per definition of u_{med} certain outputs have the same utility, namely, duplicates and elements in $\mathcal{U} \backslash D$. We use this observation to simplify the median utility definition and evaluate it only for elements in D instead of the entire universe $\mathcal{U}$.

Definition 6 (Median utility function)**.** *Let data set $D \in \mathcal{U}^n$ be sorted. The* median utility function $utility : I_D \to \mathbb{Z}$ *scores the utility of an element of D at position $i \in I_D$ as*

$$utility(i) = \begin{cases} i - \frac{n}{2} + 1 & \text{if } i < \frac{n}{2} \\ \frac{n}{2} - i & \text{else} \end{cases}.$$

First, we prove the equivalence of utility function $utility$ and u_{med} only for distinct data ($D \subseteq \mathcal{U}$) then we define gap to help with the utility computation for data sets with duplicates.

Theorem 1 (Utility equivalence)**.** *For $D \subseteq \mathcal{U}$ and index $i \in I_D$ we have*

$$u_{med}(D, x) = utility(i)$$

for $x \in [d_i, d_{i+1})$ with $i < n/2$ and $x \in (d_{i-1}, d_i]$ with $i \geq n/2$.

Proof: First, we show that all elements in $x \in [d_i, d_{i+1})$ for $i < n/2$ and $x \in (d_{i-1}, d_i]$ for $i \geq n/2$ have the same utility. The utility u_{med} of an element $x \in \mathcal{U}$ is based on a rank from the set $S_x = \{j \mid \text{rank}_D(x) \leq j \leq \text{rank}_D(x+1)\}$ according to Definition 5. For $i < n/2$, $x \geq d_i$ and $x + 1 < d_{i+1}$ we have $\text{rank}_D(x+1) = \text{rank}_D(d_{i+1})$. All elements in the open range (d_i, d_{i+1}) have the same rank set $S = \{\text{rank}_D(x+1)\}$. The rank set for d_i, S_{d_i}, is a superset of S that also includes ranks smaller than $\text{rank}_D(x+1)$. However, $\text{rank}_D(x+1) = S_{d_i} \cap S$ minimizes the term $|\text{rank}_D(x+1) - n/2|$ since it is the value closest to $n/2$. Thus, all elements in the half-open range $[d_i, d_{i+1})$ have the same utility. Analogously, for $i \geq n/2$ elements in $(d_{i-1}, d_i]$ have the same utility.

For $i \in I_D$ and sorted $D \subseteq \mathcal{U}$ we have $\text{rank}_D(d_i) = i$ and $S_{d_i} = \{\text{rank}_D(d_i), \text{rank}_D(d_i + 1)\} = \{i, i+1\}$. Thus,

$$\begin{aligned} u_{\text{med}}(D, d_i) &= -\min_{j \in \{i, i+1\}} \left| j - \frac{n}{2} \right| \\ &= \begin{cases} i + 1 - \frac{n}{2} & \text{if } i < \frac{n}{2} \\ \frac{n}{2} - i & \text{else} \end{cases} \\ &= utility(i). \end{aligned}$$

■

Thus, the sensitivity of $utility$ is the same as u_{med}. We stress that $utility(i)$ only depends on the *position* i in the sorted data. Basically, we assume all elements in D are distinct, in this case $utility(i) = u_{\text{med}}(D, d_i)$. To only retain the correct utility in the presence of duplicates we define gap next[5].

Definition 7 (Gap)**.** *The* gap function $gap : I_D \to \mathbb{N}_0$ *provides the number of consecutive elements in $\mathcal{U}$ with the same utility as d_i with*

$$gap(i) = \begin{cases} d_{i+1} - d_i & \text{if } i < \frac{n}{2} - 1 \\ 1 & \text{if } i = \frac{n}{2} - 1 \\ d_i - d_{i-1} & \text{else} \end{cases}. \tag{2}$$

Note that gap is defined for all n indices although there are only $n-1$ gaps between values in D. We set the median's gap to 1 as it is the only element not contained in the union of all half-open ranges. If D contains duplicates gap is zero for all except the duplicate closest to the median. Thus, a gap value of zero indicates incorrect utility for a duplicate and we use this to eliminate such utility values in the following.

First, with the help of $utility$ we define the unnormalized selection probability, which we call *weight*.

Definition 8 (Weight)**.** *The* weight function $weight : I_D \to \mathbb{R}$ *gives the unnormalized selection probability for an element at index $i \in I_D$ as*

$$weight(i) = \exp\left(\epsilon \cdot utility(i)\right)$$

where ϵ is the privacy parameter from Definition 1.

[5]Computation of $utility$, gap with static access is illustrated in Appendix B

TABLE I. u_{MED} COMPARED WITH $utility$ WITH STATIC ACCESS PATTERN AND gap FOR SORTED $D = \{2, 2, 6, 6, 7, 7\}, \mathcal{U} = \{1, \ldots, 10\}$. TO COVER UTILITY FOR ALL OF $\mathcal{U}$ WE ADD $\min(\mathcal{U}), \max(\mathcal{U})$ TO D.

index i	0	1	2	–	3	4	5	6	–	7
sorted D	1	2	2	3,4,5	6	6	7	7	8,9	10
$\text{rank}_D(\cdot)$	0	1	1	3	3	3	5	5	7	7
$u_{\text{med}}(D, \cdot)$	-3	-1	-1	-1	0	0	-1	-1	-3	-3
$utility(i)$	-3	-2	-1	-1	0	0	-1	-2	-3	-3
$gap(i)$	1	0	4	–	1	0	1	0	–	3

$\min(\mathcal{U}), \max(\mathcal{U})$ Missing elements $\mathcal{U} \backslash D$

Then, we use $weight$ and gap to define the probability mass of elements with the same utility, which we call $mass$.

Definition 9 (Mass)**.** *The* probability mass function $mass : I_D \to \mathbb{R}$ *at $i \in I_D$ is*

$$mass(i) = \sum_{h=0}^{i} weight(h) \cdot gap(h).$$

To ensure that $mass$ covers *all* elements in $\mathcal{U}$ we append the smallest and largest universe element to the beginning resp. end of D before computing $mass$. Now, we show that $mass$ is the (unnormalized) cumulative density function for the distribution defined by $\mathcal{M}_u^\epsilon(D)$.

Theorem 2. *Let $\mathcal{O} = \{d_0, \ldots, d_i\} \subseteq \mathcal{U}$ with D sorted, $\min(\mathcal{U})$, $\max(\mathcal{U}) \in D$ and $i \in I_D$, then*

$$\frac{mass(i)}{R} = \sum_{o \in \mathcal{O}} Pr[\mathcal{M}_u^\epsilon(D) = o],$$

with $u = u_{med}$ and normalization $R = \sum_{o' \in \mathcal{U}} Pr[\mathcal{M}_u^\epsilon(D) = o']$.

Proof: Without duplicates $utility = u_{\text{med}}$ (Theorem 1), thus, $weight(i) = \exp(\epsilon \cdot u_{\text{med}}(D, d_i))$ for $i \in I_D$. With duplicates $weight$ can produce incorrect values, however, $weight(i) \cdot gap(i) = 0$ as gap is zero for all duplicates except the one closest to the median. In other words, we eliminate weights based on incorrect utility values as they do not alter the sum $mass[i] = \sum_{h=0}^{i} weight(h) \cdot gap(h)$.

On the other hand, $gap > 0$ indicates the number of consecutive elements in $\mathcal{U}$ with same utility, and $weight(i) \cdot gap(i)$ is their unnormalized probability mass. Thus, $mass[i]$ equals the sum of unnormalized probabilities for elements in $\mathcal{O} = \{\min(\mathcal{U}), \ldots, d_i\}$, and $mass[i]/R$ equals normalized probabilities $\sum_{o \in \mathcal{O}} \Pr[\mathcal{M}_u^\epsilon(D) = o]$. ■

An example for $utility$ and gap can be found in Table I. It illustrates that $utility$ for sorted D is just a sequence that first increases, then decreases after the median. As mentioned above, we add $\min(\mathcal{U})$ to the beginning and $\max(\mathcal{U})$ to the end of D (highlighted in light gray in Table I). The utility for "missing elements" in $\mathcal{U} \backslash D$ (dark gray columns) is the same as for the preceding or succeeding element in D. Furthermore, gap is zero for the duplicates furthest away from the median and otherwise indicates the number of consecutive elements in $\mathcal{U}$ with the same utility (e.g., $gap(2) = 4$ as $2, 3, 4, 5$ have the same utility as $d_2 = 2$).

C. Median Sampling

We use *inverse transform sampling* to sample the differentially private median from the cumulative distribution function $mass$ by finding an index $j \in I_D$[6] such that $mass(j-1) \leq r < mass(j)$ for a uniform random r. Finally, we select an element at uniform random among the $gap(j)$ consecutive elements with the same utility as the element at index j. Now, with our simplified utility, we do not need to iterate over all elements in $\mathcal{U}$, but only over elements in D while still covering all "missing" elements ($\mathcal{U} \backslash D$) via gap.

D. Input Pruning with non-decreasing Utility

However, n might be large and we show how to prune D via [1] before applying our median selection. Alternative pruning approaches are used in [27, 35, 44] (see Section VII). Next, we explain pruning, define accuracy, and present the maximum pruning steps for a given accuracy.

PRUNE is a technique used by Aggarwal et al. [1] to securely find the median of two parties A, B with respective data sets D_A, D_B. We assume the data size of each party, i.e., $|D_A|, |D_B|$, to be known, however, it can be hidden via additional padding. As preprocessing, the parties A, B sort their respective data sets D_A, D_B and only retain the smallest $k = \lceil |D_A| + |D_B| \rceil / 2$ values[7]. Then, they pad the remaining data with $-\infty, +\infty$ to be of size $2^{\lceil \log_2(k) \rceil}$ in a way that preserves the position of the median (see Appendix D for details). In each pruning step the parties compute their respective medians, m_A, m_B, perform a secure comparison $m_A < m_B$, and use the result to discard the halves of their data that cannot contain their mutual median, i.e., A retains the upper half of D_A if $m_A < m_B$ and the lower half otherwise, B does the opposite. After $\log n$ iterations only their exact mutual median remains. We denote data sets D_A, D_B after pruning step s as D_A^s, D_B^s and their union as D^s. Note that PRUNE does not violate differential privacy as we only consider PRUNE-neighboring data sets with the same comparison results similar to [28]. The median m of D is also the median of D^s as shown in [1, Lemma 1]. How the data D is distributed among parties changes the intermediary outcome of the pruning, i.e., what elements remain in D_A^s, D_B^s. However, utility depends on an element's closeness to the median which remains or increases if elements in between are removed; for details we refer to Appendix C. Before we can find the maximum number of pruning steps we first define what accuracy we want to maintain after pruning. We separate the universe $\mathcal{U}$ in two disjunct sets of *remaining elements* $\mathcal{R}$ and *pruned elements* $\mathcal{P}$ where $\mathcal{R} = \{x \in \mathcal{U} \mid \min(D^s) \leq x \leq \max(D^s)\} \subseteq \mathcal{U}$ and $\mathcal{P} = \{x \in \mathcal{U} \mid x < \min(D^s) \text{ or } x > \max(D^s)\} = \mathcal{U} \backslash \mathcal{R}$. Note that $\mathcal{R}$ contains the universe elements closest to the median.

Definition 10 (Accuracy)**.** *Let $u = u_{med}$, then* accuracy *is*

$$p_{\mathcal{R}} = 1 - p_{\mathcal{P}} = \sum_{x \in \mathcal{R}} Pr[\mathcal{M}_u^{\epsilon}(D^s) = x],$$

i.e., $p_{\mathcal{R}}$ is the probability mass of all remaining elements.

[6]For notational convenience let $j - 1 < 0$ be 0.

[7]If the data contains duplicates, $\lceil \log_2 n \rceil + 1$ bits are added to the element's binary representation to make it unique, which is required for the security proof from [1, Section 3.2]. We implement the uniqueness encoding but omit it in the presented protocol to simplify its description.

With accuracy $p_{\mathcal{R}} > 0.5$ it is more likely to select the differentially private median among $\mathcal{R}$ than among $\mathcal{P}$. In our evaluation we use accuracy $p_{\mathcal{R}} = 0.9999$. The number of pruning steps s enables a trade-off between accuracy $p_{\mathcal{R}}$ and computation complexity: smaller s leads to higher accuracy and larger s translates into smaller input size for the secure computation. We are interested in the maximum number of pruning steps such that it is more likely to select an element from $\mathcal{R}$ instead of $\mathcal{P}$.

Theorem 3 (Upper Bound for Pruning Steps)**.** *Let D be a data set with data universe $\mathcal{U}$, $\epsilon > 0$, and $0 < \alpha < 1$. The upper bound for pruning steps s fulfilling $p_{\mathcal{R}} \geq \alpha$ is*

$$\lfloor \log_2(\epsilon n) - \log_2\left(\log_e\left(\frac{\alpha}{1-\alpha}(|\mathcal{U}| - 1)\right)\right) - 1 \rfloor.$$

Proof: We find the maximum number of pruning steps s by examining what the maximum probability mass $p_{\mathcal{P}}$ for pruned elements can be.

First, note that the utility for all $x \in \mathcal{P}$ is the same independent of the values in D^s: Half of the values in $\mathcal{P}$ are smaller (resp., larger) than the median m of D^s, i.e., $\text{rank}_{D^s}(x) = 0$ if $x < m$ and $\text{rank}_{D^s}(x) = |D^s|$ otherwise. Thus, $u_{\text{med}}(D^s, x) = -\left|0 - \frac{|D^s|}{2}\right| = -\left||D^s| - \frac{|D^s|}{2}\right| = -\frac{n}{2^{s+1}}$ since $|D^s| = \frac{n}{2^s}$. (Recall that D is padded before pruning such that n is a power of two.)

As the utility, and thus selection probability, is the same for all elements in $\mathcal{P}$ the probability mass $p_{\mathcal{P}}$ is maximized if $|\mathcal{P}|$ is maximized. The maximum for $|\mathcal{P}|$ is $|\mathcal{U}| - 1$ as $\mathcal{R}$ must contain at least one element, the median m.

Let $p'_{\mathcal{R}}, p'_{\mathcal{P}}$ be the unnormalized probability masses $p_{\mathcal{R}}, p_{\mathcal{P}}$ respectively, then

$$p'_{\mathcal{R}} = \exp(\epsilon u_{\text{med}}(D^s, m)) = 1$$

since $\mathcal{R} = \{m\}$ and $u_{\text{med}}(D^s, m) = 0$, and

$$p'_{\mathcal{P}} = (|\mathcal{U}| - 1) \exp\left(-\epsilon \frac{n}{2^{s+1}}\right)$$

with normalization term $R = p'_{\mathcal{P}} + p'_{\mathcal{R}}$. Now accuracy $p_{\mathcal{R}}$ of at least α is equivalent to

$$\begin{aligned} \alpha \leq \frac{p'_{\mathcal{R}}}{R} &= \frac{1}{(|\mathcal{U}| - 1)\exp\left(-\frac{\epsilon n}{2^{s+1}}\right) + 1} \\ \Leftrightarrow \exp\left(-\frac{\epsilon n}{2^{s+1}}\right) &\leq \frac{1-\alpha}{\alpha(|\mathcal{U}| - 1)} \\ \Leftrightarrow \log_e\left(\frac{\alpha(|\mathcal{U}| - 1)}{1-\alpha}\right) &\leq \frac{\epsilon n}{2^{s+1}} \\ \Leftrightarrow s \leq \log_2\left(\frac{\epsilon n}{\log_e\left(\frac{\alpha}{1-\alpha}(|\mathcal{U}| - 1)\right)}\right) - 1. \end{aligned}$$

As $s \in \mathbb{N}$ we use $s = \lfloor \log_2\left(\frac{\epsilon n}{\log_e\left(\frac{\alpha}{1-\alpha}(|\mathcal{U}|-1)\right)}\right) - 1 \rfloor$ which concludes the proof. ∎

This is a *worst-case analysis* and a tighter upper bound can be obtained by using $|\mathcal{P}|$ instead of $|\mathcal{U}| - 1$. However, the size of $\mathcal{P}$ leaks information about D, hence, we refrain from using

the tighter bound. Furthermore, we guarantee an accuracy of *at least* α, the actual accuracy can be even higher.

Lemma 1. *With $s \in \mathcal{O}(\log(n) - \log\log(|\mathcal{U}|))$ the pruned data set's size is sublinear in the size of the data universe, i.e., $|D^s| = \frac{n}{2^s} \in \mathcal{O}(\log(|\mathcal{U}|))$*[8].

V. Secure Sublinear Time Differentially Private Median Computation

We describe our full protocol in Section V-A. In Section V-B we detail optimizations and present a runtime complexity analysis in Section V-C. In Section V-D we prove the security of our protocol.

A. Protocol Description

Our protocol uses pruning developed by Aggarwal et al. [1], which requires padding as a pre-processing step as described in Appendix D. The selection probabilities are computed on securely sorted, pruned data realized via oblivious merging from Huang et al. [31], detailed in Appendix E.

The notation "$\underline{A}$:" before an operation indicates that only party A performs the following operation, likewise for party B, and $L[i]$ denotes the element at index i in array L. Our protocol has four steps, denoted with (I)–(IV).

(I): Input Pruning (Algorithm 1): Both parties prune their data sets D_A, D_B to D_A^s, D_B^s via [1] using secure comparison realized with garbled circuits.

(II): Oblivious Merge & Secret Sharing (Algorithm 2): The parties merge their pruned data D_A^s, D_B^s into sorted D^s via bitonic mergers from [31] implemented with garbled circuits. Note that $D^s = \{d_0^s, \ldots, d_{|D^s|-1}^s\}$ is secret shared, i.e., A holds shares $\langle d_i^s \rangle_A$, B holds $\langle d_i^s \rangle_B$ for all $i \in I_{D^s}$.

(III): Selection Probability (Algorithm 3): The parties compute $utility$, $weight$, and gap to produce shares of $mass$. Each party $P \in \{A, B\}$ now holds shares $\langle d_i^s \rangle_P$, $\langle gap(i) \rangle_P$ and $\langle mass(i) \rangle_P$ for all $i \in I_{D^s}$,

(IV): Median Selection (Algorithm 4): The parties reconstruct all shares and select the differentially private median via inverse transform sampling realized with garbled circuits. First, they sample $d_j^s \in D^s$ based on $mass$. Then, they select the differentially private median $\widehat{m}$ at uniform random among the $gap(j)$ consecutive elements with the same utility as d_j^s.

B. Optimizations

To optimize the performance of the secure computation we utilize garbled circuits as well as secret sharing to use their respective advantages. E.g., multiplication of two b-bit values expressed as a Boolean circuit leads to a large circuit of size $\mathcal{O}(b^2)$ and is more efficiently done via secret sharing. On the other hand, comparison is more efficient with garbled circuits. Algorithms 2, 3 are implemented with garbled circuits. In Algorithm 1 only line 6 requires garbled circuits, the rest is either data-independent or executed locally. Secret shares, denoted with $\langle \cdot \rangle$, are created in Algorithm 2, used in Algorithm 3, and recombined in Algorithm 4. Furthermore, we compute the required exponentiations in Algorithm 3 line 7 without any secure computation. Next we reiterate portions of Section IV-B but in the new context of secure computation.

Sorting via Garbled Circuits: Our utility definition requires the data to be sorted which inherently relies on comparisons. Comparisons are more efficiently implemented in binary cir-

[8] We assume $n > \log(|\mathcal{U}|)$, as otherwise we do not require pruning and our input is already sublinear in the size of the universe.

Algorithm 1 Prune prunes D_A, D_B to D_A^s, D_B^s via [1].

Input: Data D_A from A, D_B from B, pruning steps s, median rank $k = \lceil (|D_A| + |D_B|)/2 \rceil$.
Output: A has pruned data D_A^s, likewise B has D_B^s.
1: $\underline{A}$: $D_A^0 \leftarrow \text{Pad}(D_A, k, +\infty)$ `//Appendix D`
2: $\underline{B}$: $D_B^0 \leftarrow \text{Pad}(D_B, k, -\infty)$
3: **for** $i \leftarrow 0$ **to** $s-1$ **do**
4: $\quad \underline{A}$: $m_A \leftarrow$ median of D_A^i
5: $\quad \underline{B}$: $m_B \leftarrow$ median of D_B^i
6: $\quad c \leftarrow m_A < m_B$
7: $\quad \underline{A}$: $D_A^{i+1} \leftarrow$ upper half of D_A^i **if** $c = 1$ **else** lower half
8: $\quad \underline{B}$: $D_B^{i+1} \leftarrow$ lower half of D_B^i **if** $c = 1$ **else** upper half
9: **end for**

Algorithm 2 MergeAndShare merges D_A^s, D_B^s into sorted D^s via [31] and secret shares it.

Input: Pruned data D_A^s from A in ascending order, array $\langle D^s \rangle_A$ of $2|D_A^s|$ random values in $\mathbb{Z}_{2^{64}}$ from A, D_B^s from B sorted in descending order.
Output: A has secret shares $\langle D^s \rangle_A$ of sorted union of pruned data, resp. B has $\langle D^s \rangle_B$.
1: $D^s \leftarrow D_A^s$ appended with D_B^s
2: $\text{Merge}(0, |D^s| - 1, D^s)$ `//Appendix E`
3: $\langle D^s \rangle_B \leftarrow D^s - \langle D^s \rangle_A \mod 2^{64}$
4: **return** $\langle D^s \rangle_B$ **to** B

Algorithm 3 SelectionProbability computes the probabilities for the median utility.

Input: Secret shares $\langle D^s \rangle_A$ from A, resp. $\langle D^s \rangle_B$ from B, of the sorted data D^s, and number k of nonces.
Output: A holds secret shares $\langle gap \rangle_A$ of gaps and $\langle mass \rangle_A$ of probability masses, also nonces N_A^1, N_A^2; likewise party B has $\langle gap \rangle_B, \langle mass \rangle_B, N_B^1, N_B^2$.
1: $\underline{A}$: $\langle D^s \rangle_A \leftarrow (0, \langle D^s \rangle_A, 0)$
2: $\underline{B}$: $\langle D^s \rangle_B \leftarrow (\min(\mathcal{U}), \langle D^s \rangle_B, \max(\mathcal{U}))$
3: **each** party $P \in \{A, B\}$ **does**
4: $\quad$ Define arrays $\langle mass \rangle_P, \langle gap \rangle_P$ of size $|D^s|$
5: $\quad$ **for** $i \leftarrow 0$ **to** $|D^s| - 1$ **do**
6: $\quad\quad utility \leftarrow \begin{cases} i - \frac{|D^s|}{2} + 1 & \textbf{if } i < \frac{|D^s|}{2} \\ \frac{|D^s|}{2} - i & \textbf{else} \end{cases}$
7: $\quad\quad weight \leftarrow \exp(\epsilon \cdot utility)$
8: $\quad\quad \langle gap[i] \rangle_P \leftarrow \begin{cases} \langle d_{i+1}^s \rangle_P - \langle d_i^s \rangle_P & \textbf{if } i < \frac{|D^s|}{2} - 1 \\ \langle 1 \rangle_P & \textbf{if } i = \frac{|D^s|}{2} - 1 \\ \langle d_i^s \rangle_P - \langle d_{i-1}^s \rangle_P & \textbf{else} \end{cases}$
9: $\quad\quad t \leftarrow \langle mass[i-1] \rangle_P$ **if** $i > 0$ **else** 0
10: $\quad\quad \langle mass[i] \rangle_P \leftarrow t + weight \cdot \langle gap[i] \rangle_P$
11: $\quad$ **end for**
12: $\quad$ Draw lists of k nonces N_P^1, N_P^2 from $[0, \max(\mathcal{U}) - \min(\mathcal{U})]$
13: **end each**

Algorithm 4 MEDIANSELECTION selects the median via inverse transform sampling.

Input: Secret shares $\langle gap \rangle_A$ of gaps, $\langle mass \rangle_A$ of probability masses, and $\langle D^s \rangle_A$ of A's (pruned) data, also lists of nonces N_A^1, N_A^2 from A; resp. $\langle gap \rangle_B, \langle mass \rangle_B, \langle D^s \rangle_B, N_B^1, N_B^2$ from B.
Output: Differentially private median $\widehat{m}$ of $D_A \cup D_B$.

1: $R \leftarrow \langle mass\,[|D^s|-1] \rangle_A + \langle mass\,[|D^s|-1] \rangle_B \mod 2^{64}$
2: $r \leftarrow$ RANDOMDRAW$(R+1, N_A^1, N_B^1)$ `//Appendix F`
3: Initialize $j \leftarrow -1$ and define d, g
4: **for** $i \leftarrow 0$ **to** $|D^s|-1$ **do**
5: $\quad e \leftarrow \langle d_i^s \rangle_A + \langle d_i^s \rangle_B \mod 2^{64}$ `//Recombine shares`
6: $\quad gap \leftarrow \langle gap[i] \rangle_A + \langle gap[i] \rangle_B \mod 2^{64}$
7: $\quad mass \leftarrow \langle mass[i] \rangle_A + \langle mass[i] \rangle_B \mod 2^{64}$
8: $\quad$ **if** $r < mass$ **and** $j = -1$ **then**
9: $\quad\quad d \leftarrow e$; $g \leftarrow gap$; $j \leftarrow i$
10: $\quad$ **end if**
11: **end for**
12: $x \leftarrow$ RANDOMDRAW(g, N_A^2, N_B^2)
13: $\widehat{m} \leftarrow \begin{cases} d+x & \textbf{if } j < \frac{|D^s|}{2} - 1 \\ d-x & \textbf{else} \end{cases}$
14: **return** $\widehat{m}$ **to** A, B

cuits than arithmetic circuits, hence, we use the former. We leverage that D_A^s and D_B^s are already sorted and merge them instead of sorting the union. Oblivious merging of two lists of n sorted b-bit elements only requires $2bn\log(n)$ binary gates whereas oblivious sorting requires $\Theta(n\log(n))$ with a large constant factor [31]. We use *bitonic mergers* from Huang et al. [31] which require a bitonic list as input, i.e., a list that monotonically increases then decreases (or vice versa). We can generate a bitonic list by appending D_A^s sorted in ascending order with D_B^s sorted in descending order (Algorithm 2 line 1).

Exponentiation without Secure Computation: To compute the probabilities for $i \in I_{D^s}$ we require exponentiations of the form $\exp(\epsilon \cdot utility(i))$. Note that none of the arguments are secret, since ϵ is a public parameter and we defined $utility$ to not require data access. Therefore, we are able to compute the required exponentiations without any secure computation.

Addition and Multiplication via Secret Sharing: We want to compute the probability mass $weight(i) \cdot gap(i)$ which requires two operations: subtractions over secret data D^s to compute gap and multiplication of public values ($weight$), with secret values (gap). Both operations are more efficiently implemented with secret sharing.

Selection via Garbled Circuits: The median selection is realized with inverse transform sampling which is better suited for garbled circuits as it requires comparisons. We draw a random r via nonces (see Appendix F) and compute the first index $j \in I_{D^s}$ such that the probability mass is larger than r: $mass(j) > r$ (line 8 in Algorithm 4). Note that we do not sample r from $[0,1]$ but from $[0,R]$ where $R = mass(|D^s|-1)$, i.e., the normalization factor from Equation (1). This allows us to use the unnormalized probabilities and eliminates divisions used in normalization. In the final step, we select the differentially private median at uniform random among the $gap(j)$ consecutive elements with the same utility (and thus probability) as d_j^s (line 13 in Algorithm 4).

C. Runtime Complexity Analysis

Step (I), requires $s \in \mathcal{O}(\log n - \log\log|\mathcal{U}|)$ comparisons (see Theorem 3). Step (II) requires $2b|D^s|\log|D^s|$ binary gates [31] for $|D^s|$ elements with bit length b. Steps (III) and (IV) require $\mathcal{O}(|D^s|)$ operations each. Since $|D^s| \in \mathcal{O}(\log|\mathcal{U}|)$ (Lemma 1), our overall runtime is $\mathcal{O}(\max\{\log n - \log\log|\mathcal{U}|, \log|\mathcal{U}| \cdot \log\log|\mathcal{U}|\})$, which is sublinear in n for $n > \log|\mathcal{U}|^{\log|\mathcal{U}|+1}$, and sublinear in $|\mathcal{U}|$ otherwise.

D. Security

We combine different secure computation techniques in the *semi-honest model* introduced by [24] 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 secure computation. To analyze the security of the entire protocol we rely on the well-known *composition theorem* [24, Section 7.3.1]. Basically, a secure protocol that uses an ideal functionality (a subroutine provided by a trusted third party) remains secure if the ideal functionality is replaced with a secure computation implementing the same functionality. We consider PRUNE-neighboring data sets (Definition 3), i.e., neighboring data sets with the same pruning result.

Theorem 4 (Security)**.** *Our protocol securely implements the ideal functionality of differentially private median selection via the steps* PRUNE*,* MERGEANDSHARE*,* SELECTIONPROBABILITY *and* MEDIANSELECTION *in the semi-honest model.*

Proof: We use the composition theorem to analyze the security of our protocol: We define required ideal functionalities, show how they map to our garbled circuit implementation (steps (I), (II), (IV)), and how it combines with secret sharing (step (III)). Aggarwal et al. [1] developed the input pruning we utilize and give a simulation-based security proof only using comparisons as ideal functionality. PRUNE, a partial execution of [1], allows the same simulation argument (see Appendix G). Note that these comparisons leak nothing about PRUNE-neighboring data sets. For the interactive computation we require the following ideal functionalities:

- $c \leftarrow$ SECURECOMPARE$^{\text{ideal}}(m_A; m_B)$.
  In step (I) the ideal functionality on input m_A, m_B, i.e., median from A, B respectively, outputs the result of comparison $m_A < m_B$ as bit c to both parties.
- $\langle D^s \rangle_A, \langle D^s \rangle_B \leftarrow$ MERGEANDSHARE$^{\text{ideal}}(D_A^s; D_B^s)$.
  In step (II) the ideal functionality receives as input the pruned data D_A^s, D_B^s from A, B respectively, and outputs the sorted, merged data as secret shares, i.e., $\langle D^s \rangle_A, \langle D^s \rangle_B$ is output to A, B respectively.
- $\widehat{m} \leftarrow$ MEDIANSELECTION$^{\text{ideal}}(\langle gap \rangle_A, \langle mass \rangle_A, \langle D^s \rangle_A; \langle gap \rangle_B, \langle mass \rangle_B, \langle D^s \rangle_B)$.
  In step (IV) party A inputs $\langle gap \rangle_A, \langle mass \rangle_A, \langle D^s \rangle_A$, party B inputs $\langle gap \rangle_B, \langle mass \rangle_B, \langle D^s \rangle_B$ and the ideal functionality outputs the DP median $\widehat{m}$ to both.

Step (III), SELECTIONPROBABILITY, performs local computations without interaction, and does not require any ideal functionality. We realize SECURECOMPARE$^{\text{ideal}}$ with garbled circuits in Algorithm 1 line 6. The ideal functionality

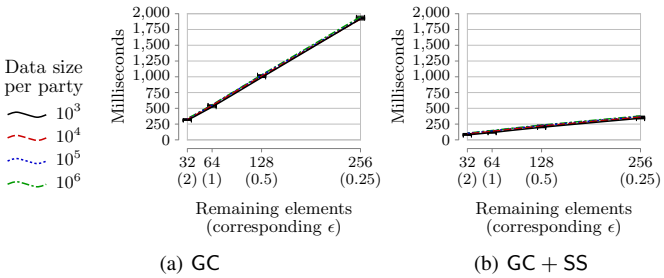
MERGEANDSHARE$^{\text{ideal}}$, from merging step (II), is implemented as MERGEANDSHARE in Algorithm 2 with garbled circuits. Note that A provides the randomness for the secret sharing, i.e., $\langle D^s \rangle_A$ as additional input which is not required by the ideal functionality. Garbled circuits are also used in the selection step (IV), where MEDIANSELECTION$^{\text{ideal}}$ is implemented as MEDIANSELECTION in Algorithm 4. Additionally, to the input mentioned for the ideal functionality, the parties also provide nonces as a source of randomness. We rely on the established security proofs for garbled circuits in the semi-honest model provided by Lindell and Pinkas [36]. Outputs of (II), (III) are intermediate states of our interactive computation. As noted in [24, Section 7.1.2.3] such state can be maintained securely among the computation parties in a secret sharing manner. For security proofs of secret sharing we refer to [46] and for security proofs for converting between garbled circuits and secret sharing we refer to [14]. Altogether, the execution of PRUNE$^{\text{ideal}}$, MERGEANDSHARE$^{\text{ideal}}$, SELECTIONPROBABILITY, and MEDIANSELECTION$^{\text{ideal}}$ constitute the ideal functionality for differentially private median. Utilizing the composition theorem and [24, Section 7.1.2.3] we replace the ideal functionality with secure implementations PRUNE, MERGEANDSHARE, MEDIANSELECTION and secret share the intermediate states. ■

VI. EVALUATION

Our implementation is written in C/C++ using the mixed-protocol framework *ABY* developed by Demmler et al. [14]. We chose ABY as it supports secure two-party computation based on arithmetic sharing and Yao's garbled circuits and provides efficient conversion between them. We implemented two versions of our protocol – GC, with garbled circuits, and GC + SS, with garbled circuits as well as secret sharing – to show that using a mixed-protocol, which requires additional conversion between the schemes, is still more efficient than only utilizing garbled circuits. For evaluation we used the Open Payments 2017 data set from the Centers for Medicare & Medicaid Services (CMS) [11]. The CMS collects all payments made to physicians from drug or medical device manufacturers as required by the Physician Payments Sunshine Act. We evaluated different numbers of remaining elements after pruning (i.e., different sizes of D^s) which is inversely proportional to the privacy parameter ϵ as the number of pruning steps depends on it (see Theorem 3). We used an accuracy value of 0.9999 to determine the number of pruning steps. We ran the evaluation on AWS t2.medium instances with 2GB RAM and 4 vCPUs (where vCPU count roughly translates to thread count). As garbled circuits and pruning are interactive protocols they are influenced by network delay and bandwidth, therefore, we evaluated our protocol in real networks between different AWS regions with round trip times (RTT) of none (LAN), 12 ms (Ohio–N. Virginia), 25 ms (Ohio–Canada), and 100 ms (Ohio–Frankfurt), with bandwidths of 1 GBits/s, 430 MBits/s, 160 MBits/s and 100 MBits/s respectively.

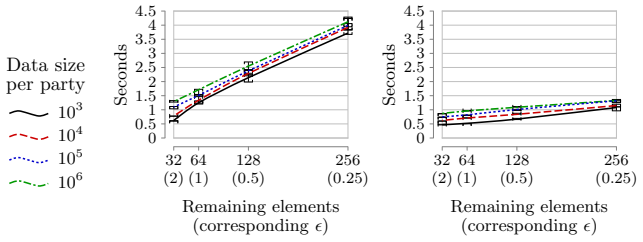
A. Runtime

We evaluated the runtime of GC and GC + SS, which includes setup time (OT extensions, garbling) and online time in seconds (or milliseconds in the LAN setting). The runtime evaluations with increasing delays and decreasing bandwidths



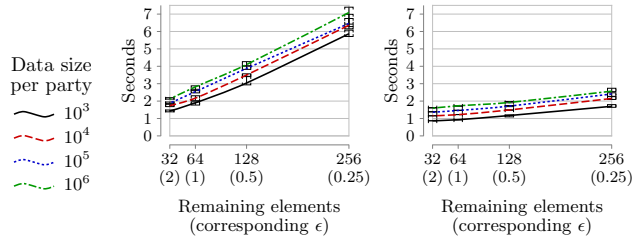
(a) GC (b) GC + SS

Fig. 4. Runtime without network delay and 1 GBits/s bandwidth (LAN).



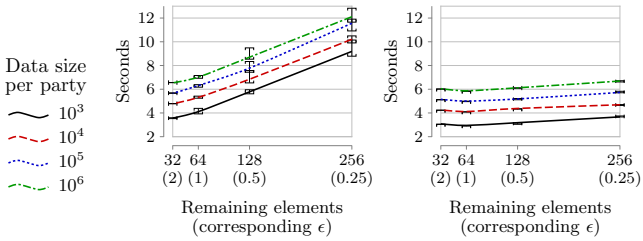
(a) GC (b) GC + SS

Fig. 5. Runtime for ∼12 ms RTT, ∼430 MBits/s (Ohio and N. Virginia).



(a) GC (b) GC + SS

Fig. 6. Runtime for ∼25 ms RTT, ∼160 MBits/s (Ohio and Canada).



(a) GC (b) GC + SS

Fig. 7. Runtime for ∼100 ms RTT, ∼100 MBits/s (Ohio and Frankfurt).

are presented in Fig. 4–7. In each figure we plotted different data set sizes $|D_A| = |D_B| = |D|/2 \in \{10^3, 10^4, 10^5, 10^6\}$ to show that our protocol scales with increasingly larger data sets. The runtime is the median of 20 runs and the 25$^{\text{th}}$- as well as 75$^{\text{th}}$-percentile are indicated with brackets. The runtime plots for GC and GC + SS have the same scale (and are grouped side-by-side) to allow for an easier comparison between the two. The advantage of GC + SS over GC is most obvious in the LAN setting, where the runtime for GC + SS, see Fig. 4b, is always below that of GC, see Fig. 4a. The

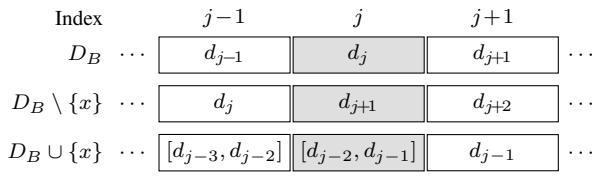


Fig. 8. Neighbors of D_B in relation to comparison index j used by PRUNE (values highlighted in gray). Neighbors are D_B with a value $x \in D_B$ removed or $x \in \mathcal{U}$ added, illustrated for $x < d_j$. All data sets are sorted.

same is true for modest network delay as can be seen by comparing Fig. 5b with Fig. 5a. For network delay of up to 100 ms with 100 MBits/s bandwidth GC + SS is still faster than GC but less so for 32 remaining elements ($\epsilon = 2$), as shown in Fig. 6 and 7. The reason for GC + SS being not much faster is the increased number of interactive pruning steps required to reach this number of remaining elements. Also, the number of additional garbled circuits to go from GC + SS to GC is smaller for few remaining elements (see Fig. 10a), so that the pruning has more impact. Even for millions of records GC + SS has a runtime of less than 2.6 seconds with 25 ms network delay (Fig. 6b) and less than 7 seconds for 100 ms delay (Fig. 7b).

B. PRUNE-*neighboring*

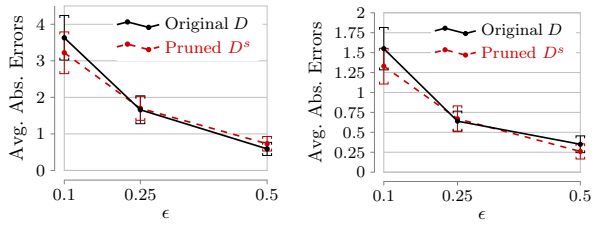
Recall, PRUNE compares the *sorted, padded* data D_A, D_B at some fixed index j in each pruning step, and a neighbor is D_B with an element x removed or added. As Fig. 8 illustrates, comparing a neighbor at index j is similar to using the original D at an adjacent index. Thus, neighbors are likely PRUNE-neighbors when the data contains multiple duplicates or is dense (no large gaps between values) and less so for sparse, unique data. In more detail, we first consider $x < d_j$ where d_j denotes the value of D_B at index j. Let the data be padded to some fixed size. Then, removing x from D_B "shifts" values larger than x to the left whereas adding x can shift smaller values to the right in the sorted data. Removing $x \in D_B$ leads to a single shift left, i.e., PRUNE uses d_{j+1} instead of d_j. For addition at most two right shifts can occur as we now have to consider $x \in \mathcal{U}$ instead of $x \in D_B$. Adding $x \in [d_{j-2}, d_{j-1}]$ places it at index j in the sorted neighbor. Thus, in the worst-case for addition, PRUNE uses d_{j-2} instead of d_j. Note that adding/removing $x \geq d_j$ affects only positions larger than j, and all such neighbors are PRUNE-neighbors for this index. Also, if the original comparison (of D_A, D_B at j) is true, then removing $x < d_j$ produces the same result in PRUNE (neighbor has an even larger value at j). Likewise if it is false and we add x. To empirically verify that PRUNE-neighboring (Definition 3) is not too restrictive we evaluated multiple columns from real-world data sets [11, 33, 51, 54], and found that all neighbors are also PRUNE-neighbors. To illustrate our evaluation methodology one can imagine the neighboring definition in differential privacy (DP) as a graph. Each database is a vertex and if two data sets are neighbors they are connected by an edge. The common neighboring definition in DP (adding/removing one element) results in a graph. PRUNE-neighboring is a restriction on that graph in the sense that it removes certain edges, similar constraints on the input databases are considered in [4, 28]. Any neighboring database considered in our IND-CDP-2PC security definition must be in a connected component of the neighboring graph where all nodes have the same output of the PRUNE-function. The result of the PRUNE steps in our protocol determines the connected component the other party's database is DP in. In that sense DP with PRUNE-neighboring cannot be violated by any adversary. Any choice of inputs by party A will lead to one (but different) connected component for the DP of B's database, i.e., B's database will always remain differentially private. We empirically showed that PRUNE-neighboring is not too restrictive, i.e., it does not remove too many edges and make the resulting connected component too small. We sampled edges from the neighboring graph resulting from the common definition on real-world data sets [11, 33, 51, 54] using the following method: Given a real-world database for B, an element to be added or removed chosen by A (note that A must choose before knowing the result), and a step in the protocol does there exist any neighbor for B's database that is excluded by the PRUNE-neighboring definition. For up to 16 consecutive pruning steps (the maximum according to Theorem 3 for our highest evaluated parameters $\epsilon = 2$, and accuracy of 0.9999), we found none. Given that the connectivity in the neighboring graph is high, this implies that the connected component is expected to remain large.

TABLE II. **MINIMUM CHANGES** (WORST-CASE) IN D_B TO SAMPLE A NEIGHBOR THAT IS NOT A PRUNE-NEIGHBOR W.R.T. D_A. EVALUATED FOR 52 000 NEIGHBORS (ALL COMBINATIONS OF UP TO 50 REMOVALS AND 50 ADDITIONS WITH 20 SAMPLES PER COMBINATION). EACH ROW SHOWS THE MINIMUM CHANGES FOR $\epsilon = 1 \mid \epsilon = 2$ AND $\hat{100}$ INDICATES NONE WERE FOUND FOR UP TO 100 CHANGES.

D_A \ D_B	*Wages [51]*	*Transactions [54]*	*Times [54]*	*Payments [11]*	*Weights [33]*	*Quantities [33]*
Wages [51]	$\hat{100}$ | 18	$\hat{100}$ | 14	12 | 12	22 | 22	$\hat{100}$ | 12	46 | 21
Transactions [54]	65 | 65	8 | 8	$\hat{100}$ | 20	37 | 30	36 | 36	23 | 23
Times [54]	$\hat{100}$ | 22	33 | 18	6 | 6	$\hat{100}$ | 13	$\hat{100}$ | 21	25 | 25
Payments [11]	28 | 28	$\hat{100}$ | 11	$\hat{100}$ | $\hat{100}$	6 | 6	$\hat{100}$ | 41	$\hat{100}$ | 13
Weights [33]	$\hat{100}$ | 43	34 | 33	4 | 4	33 | 33	$\hat{100}$ | 21	48 | 19
Quantities [33]	30 | 30	$\hat{100}$ | 25	$\hat{100}$ | 12	$\hat{100}$ | 9	14 | 14	14 | 14

Group privacy extends the neighboring definition from including (or excluding) a single value to multiple values. Therefore, to quantify group privacy we consider *multiple* changes and provide a worst-case analysis for PRUNE-neighboring: Table II shows the *minimum* changes required to produce a neighbor that is not also a PRUNE-neighbor[9]. We evaluated 52 000 neighbors (all combinations of up to 50 removals and 50 additions with 20 samples per combination) for each of the 36 ways to distribute the data between two parties (6 data sets from [11, 33, 51, 54] distributed between 2 parties). PRUNE-neighboring provides only limited group privacy for the largest number of pruning steps ($\epsilon = 2$). However, for our strongest privacy guarantee $\epsilon = 0.25$ we found changes leading to violations in only 2 from 36 data set combinations, requiring at least 12 changes. Note that this is a worst-case analysis, and an average-case is provided in Appendix H. Also, sequential composition is still supported as the result of our protocol is the median selected by the exponential mechanism which can be used as input for another (DP) mechanism. (Parallel composition, running our protocol on multiple subsets of the data at once, outputs multiple median values of these subsets.)

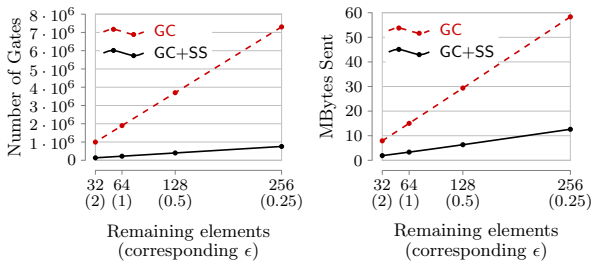
[9]Some values are the same for $\epsilon \in \{1, 2\}$ as we only report the minimum number of changes over all pruning steps.



(a) Credit card data [54], first 10^5 payment records in Cents.

(b) Walmart supply chain data [33], 175k shipment weights as integers.

Fig. 9. Absolute error averaged over 100 runs with and without pruning.



(a) Number of garbled circuit gates.

(b) Total megabytes sent.

Fig. 10. Circuit size and communication for GC vs. GC + SS.

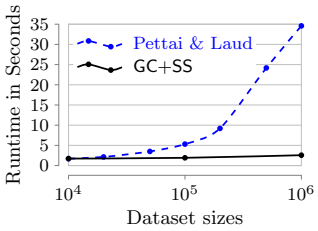


Fig. 11. Runtime of GC + SS (∼25 ms RTT and ∼160 MBits/s, 256 remaining elements, $\epsilon = 0.25$) vs. Pettai and Laud [44] (LAN).

C. Precision & Absolute Error

Our implementation uses fixed-point numbers (see Section III). As probabilities are floating point numbers we evaluated the loss of decimal precision of our secure implementation compared to a floating point operation with access to unprotected data [11]. For the maximum evaluated number of remaining elements, i.e., 256 (corresponding to $\epsilon = 0.25$), the difference for all elements combined was less than $6.5 \cdot 10^{-15}$.

Pruning preserves the elements closest to the median and the absolute error compared to the original data is small. We evaluated the absolute error, i.e., actual median versus DP median, for the exponential mechanism on original data and pruned data: Fig. 9 shows the average over 100 runs, where brackets indicate the 95% confidence interval. Before pruning the data was randomly split between both parties. Our evaluation shows the absolute error decreases by 3% on average over all evaluated $\epsilon \in \{0.1, 0.25, 0.5\}$. However, this is within the margin of error, since the confidence intervals for pruned data overlap with original data's confidence intervals.

D. Circuit size & Communication

We only report circuit size and communication for 10^6 records as smaller data sizes (i.e., fewer pruning steps) do not noticeably reduce the numbers (recall, a pruning step consists of a single comparison). The number of garbled gates for GC and GC + SS depends on the number of remaining elements and is visualized in Fig. 10a. GC requires an order of magnitude more gates as GC + SS since GC requires larger circuits for arithmetic operations whereas GC + SS avoids the need for this additional circuit complexity. The communication cost, measured in megabytes per number of remaining elements, can be found in Fig. 10b. We do not distinguish between (precomputed) setup and online phase and present the total number of megabytes sent. Whereas GC sends about 15 megabytes for 64 remaining elements ($\epsilon = 1$), GC + SS requires less than that even for 256 remaining elements ($\epsilon = 0.25$) as fewer gates have to be garbled and evaluated.

E. Comparison to Related Work

Pettai and Laud [44] compute differentially private analytics on distributed data via secret sharing for three parties, whereas we optimize our protocol for rank-based statistics of two parties and also use garbled circuits.[10] Both parties learn the PRUNE-neighborhood (for large data sets requiring pruning), but the median output can be shared (or output to only a single party) and processed further. Pettai and Laud evaluated their median computation with 48GB RAM and a 12-core 3GHz CPU in a LAN. We, on the other hand, used a comparatively modest setup (t2.medium instances with 2GB RAM, 4vCPUs) and evaluated in multiple WANs. A comparison of our protocol (with ∼25 ms delay, ∼160 MBits/s) and [44] (in a LAN) is visualized in Fig. 11. Their median computation requires 34.5 seconds for 10^6 elements in a LAN. Our protocol runs in less than 2.6 seconds with twice as many elements even with network delay and restricted bandwidth.

[10]Note that 3-party computation on secret shares are usually faster than cryptographic 2-party computations [2].

VII. RELATED WORK

We describe related work combining secure computation with differential privacy, outline alternatives to reduce the size of the data universe, and discuss other work that computes the differentially private (DP) median.

Secure Computation and DP: Dwork et al. [19] first mentioned that differential privacy combines well with secure computation. E.g., secure computation of DP sums is easily achieved via additive noise (see [25] for an overview). It was shown in [26] that some distributed DP protocols (e.g., XOR computation) can only achieve optimal accuracy when combined with secure computation. We utilize the iterative pruning from Aggarwal et al. [1] as it is a basis for more efficient secure computation protocols as shown in [49]. (Not all protocols can utilize this approach, e.g., it is not applicable when only one party learns the output [10]). Naor et al. [42] use secure two-party computation to find differentially private heavy hitters (e.g., to blacklist frequently used passwords) in the local model. They also consider malicious adversaries that try to skew the frequency. We, on the other hand, simulate the more accurate central model in the local model to find the DP median in the semi-honest model. For functions that are not

robust to potentially large noise, e.g., the median, a specific value from a data universe, the exponential mechanism, developed by McSherry and Talwar [39], is the better choice [35]. The exponential mechanism defines a probability distribution over all possible output values. Eigner et al. [21] implement the exponential mechanism in secure multiparty computation for semi-honest and malicious parties. However, they are linear in the size of the data universe: 3 semi-honest parties require 42.3 seconds to sample a universe of size 5 in a LAN on a machine with 32GB RAM and 3.2GHz. Our protocol is sublinear in the size of the data universe, requiring less than 500 milliseconds for millions of elements in a LAN with less powerful hardware (see Fig. 4b). Efficiently sampling the distribution defined by the exponential mechanism is non-trivial [18], thus, a reduction of the sampling space is considered by [6, 27, 35, 44].

Pruning and Reduction: Gupta et al. [27] suggest pruning the set of outputs for combinatorial problems from exponential to polynomial size and sample it with the exponential mechanism. We follow a different approach based on [1]. Another technique divides $\mathcal{U}$ into equal-sized ranges, selects a range with the exponential mechanism and samples a range element at uniform random [35]. However, any element in the selected range is equally likely to be output independent of its utility. Our protocol samples the median only among elements with the same utility which is exponentially more likely to select elements closer to the actual median. Pettai and Laud [44] define algorithms for privacy-preserving analytics. They securely compute the DP median with three parties but chose not to optimize their computation for the exponential mechanism and instead use the sample-and-aggregate mechanism [43]. The sample-and-aggregate mechanism divides the output in multiple equal-sized ranges, selects from each range the element closest to the median and returns a noisy average of these elements. However, the exponential mechanism, which we securely implement for the median utility function, selects an actual universe element and not a noisy approximation. The authors of [44] also apply input pruning and replace half of the excluded values with a small (resp. large) constant. They mention that this does not always preserve the median. Blocki et al. [6] use a relaxed exponential mechanism to sample a DP password frequency list in the central model. They allow a negligible error δ, i.e., they only sample the exponential mechanism correctly with probability $1-\delta$, which improves sampling from (potentially) exponential time to $\mathcal{O}(|D|^{1.5}/\epsilon)$. However, they require full access to the data D in clear.

Differentially Private Median: As mentioned above, Pettai and Laud [44] also securely compute the DP median. Their work is more general, supporting multiple DP statistics over secret-shared data, whereas we optimized our protocol for rank-based DP statistics (e.g., p^{th}-percentile, median) in a two-party setting without powerful hardware. Their protocol requires 34.5 seconds for a data size of 10^6 in a LAN [44, Fig. 1] whereas our protocol runs in less than 500 ms with twice as many elements in a LAN (Fig. 4b) and is still 13 times faster in a WAN as [44] in a LAN (Fig. 11). Median computation has also been considered in the DP query framework PINQ, developed by McSherry [40], which requires a trusted third party. Smooth sensitivity, presented in [43], analyzes the data to provide instance-specific additive noise. Yet, when smooth sensitivity is high, it still provides less accuracy than the exponential mechanism (see Section II). Also, computing the exact sensitivity itself is not trivial and requires access to the entire, sensitive data set. Another approach from Dwork and Lei [17] considers the statistical setting, where data are actually i.i.d. samples from a distribution. Their approach requires additive noise proportional to the scale of the data (approximated via interquartile range), i.e., potentially large noise, whereas our result is independent of the scale. Smith et al. [50] compute the DP median in the local model and achieve optimal error bounds without relying on secure computation and even avoid interaction; however, the local model's accuracy is limited compared to the central model ($\Omega(\sqrt{n})$ vs. $\mathcal{O}(1)$ for n parties [30]). They approximate for each party the count of elements in all subintervals of a range, structured as nodes in a tree. A server combines these noisy counts to learn the DP median. Hsu et al. [30] consider approximate counts for heavy hitters and say an algorithm is α-accurate if the returned universe element occurs with frequency that differs at most by an additive α from the true heavy hitter. They show that the lower bound for accuracy in the local model (the setting of [50]) is $\Omega(\sqrt{n})$ for n parties, whereas the central model, which we simulate via secure computation of the exponential mechanism, can achieve $\mathcal{O}(1)$ accuracy. The authors of [50] note that *general* techniques combining secure computation and differential privacy suffer from bandwidth and liveness constraints, rendering them impractical for large data sets. Our protocol shows that *specially crafted* protocols, combining different techniques and optimizations, achieve performance numbers suitable for practical applications.

VIII. Conclusion

We presented a protocol for secure differentially private median computation on private data sets from two parties with a runtime sublinear in the size of the data universe. Our protocol implements the exponential mechanism as in the local model using a distributed, secure computation protocol to achieve accuracy as in the central model without trusting a third party. For the median the exponential mechanism provides the best utility vs. privacy trade-off for low ϵ compared to additive noise (see Section II). The output is selected with an exponential bias towards elements close to the median while providing differential privacy for the individuals contained in the sensitive data. We note that our protocol can be easily extended to compute differentially private rank-based statistics such as p^{th}-percentile and interquartile range. Our experiments evaluate real-world delay and bandwidth, unlike related work [44], which we still outperform by at least a factor of 13 (with 25 ms delay and less powerful hardware) by utilizing secret sharing as well as garbled circuits for their respective advantages. We optimize our protocol by computing as little as possible using cryptographic protocols and by applying dynamic programming with a static, i.e., data-independent, access pattern, yielding lower complexity of the secure computation circuit. Our comprehensive evaluation with a large real-world payment data set [11] achieves the same high accuracy as in the central model and a practical runtime of less than 7 seconds for millions of records in real-world WANs.

Acknowledgement

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

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas, "Secure computation of the median (and other elements of specified ranks)," *Journal of cryptology*, 2010.

[2] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and performance of programmable secure computation," *IEEE security & privacy*, 2016.

[3] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the annual ACM conference on Computer and Communications Security*, ser. CCS, 2012.

[4] A. J. Biega, R. Saha Roy, and G. Weikum, "Privacy through solidarity: A user-utility-preserving framework to counter profiling," in *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017.

[5] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *Proceedings of the Symposium on Operating Systems Principles*, ser. SOSP, 2017.

[6] J. Blocki, A. Datta, and J. Bonneau, "Differentially private password frequency lists." in *Network and Distributed Systems Security Symposium*, ser. NDSS, 2016.

[7] Bloomberg News. (2018) Google and mastercard cut a secret ad deal to track retail sales. [Online]. Available: https://www.bloomberg.com/news/articles/2018-08-30/google-and-mastercard-cut-a-secret-ad-deal-to-track-retail-sales

[8] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, "How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation," in *International Conference on Financial Cryptography and Data Security*, ser. FC.

[9] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, "Students and taxes: a privacy-preserving study using secure computation," 2016.

[10] E. Boyle, Y. Ishai, and A. Polychroniadou, "Limits of practical sublinear secure computation," in *Annual International Cryptology Conference*, 2018.

[11] Centers for Medicare & Medicaid Services. (2017) Complete 2017 program year open payments dataset. [Online]. Available: https://www.cms.gov/OpenPayments/Explore-the-Data/Dataset-Downloads.html

[12] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, "Distributed differential privacy via shuffling," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT, 2019.

[13] R. A. DeMillo, D. Dobkin, and R. J. Lipton, "Even data bases that lie can be compromised," *IEEE Transactions on Software Engineering*, 1978.

[14] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *Network and Distributed Systems Security Symposium*, ser. NDSS, 2015.

[15] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Advances in Neural Information Processing Systems*, ser. NIPS, 2017.

[16] C. Dwork, "Differential privacy," in *International Colloquium on Automata, Languages, and Programming*, ser. ICALP, 2006.

[17] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *Proceedings of the annual ACM symposium on Theory of Computing*, ser. STOC, 2009.

[18] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, 2014.

[19] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation." in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT, 2006.

[20] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography Conference*, ser. TCC, 2006.

[21] F. Eigner, A. Kate, M. Maffei, F. Pampaloni, and I. Pryvalov, "Differentially private data aggregation with optimal utility," in *Proceedings of the Annual Computer Security Applications Conference*, ser. ASAC, 2014.

[22] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the annual ACM conference on computer and communications security*, ser. CCS, 2014.

[23] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, 1985.

[24] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*, 2009.

[25] S. Goryczka and L. Xiong, "A comprehensive comparison of multiparty secure additions with differential privacy," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[26] V. Goyal, D. Khurana, I. Mironov, O. Pandey, and A. Sahai, "Do distributed differentially-private protocols require oblivious transfer?" in *Leibniz International Proceedings in Informatics*, ser. LIPIcs, 2016.

[27] A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar, "Differentially private combinatorial optimization," in *Proceedings of the annual ACM SIAM symposium on Discrete Algorithms*, ser. SODA, 2010.

[28] X. He, A. Machanavajjhala, C. Flynn, and D. 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*, ser. CCS, 2017.

[29] N. Holohan, D. J. Leith, and O. Mason, "Optimal differentially private mechanisms for randomised response," *IEEE Transactions on Information Forensics and Security*, 2017.

[30] J. Hsu, S. Khanna, and A. Roth, "Distributed private heavy hitters," in *International Colloquium on Automata, Languages, and Programming*, ser. ICALP, 2012.

[31] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *Network and Distributed Systems Security Symposium*, ser. NDSS, 2012.

[32] M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan, and M. Yung, "Private intersection-sum protocol with applications to attributing aggregate ad conversions," Cryptology ePrint Archive, Report 2017/738, 2017, https://eprint.iacr.org/2017/738.

[33] Kaggle.com, "Walmart supply chain: Import and

shipment," https://www.kaggle.com/sunilp/walmart-supply-chain-data/data, 2018, Retrieved: January 29, 2019.
[34] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?" *SIAM Journal on Computing*, 2011.
[35] N. Li, M. Lyu, D. Su, and W. Yang, "Differential privacy: From theory to practice," *Synthesis Lectures on Information Security, Privacy, & Trust*, 2016.
[36] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *Journal of Cryptology*, 2009.
[37] D. Mazieres and D. Miller, "Source of arc4random.c." [Online]. Available: https://opensource.apple.com/source/Libc/Libc-1158.50.2/gen/FreeBSD/arc4random.c
[38] A. McGregor, I. Mironov, T. Pitassi, O. Reingold, K. Talwar, and S. Vadhan, "The limits of two-party differential privacy," in *Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS, 2010.
[39] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS, 2007.
[40] F. D. 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*, ser. SIGMOD, 2009.
[41] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *Annual International Cryptology Conference*, 2009.
[42] M. Naor, B. Pinkas, and E. Ronen, "How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior," in *Proceedings of the annual ACM conference on computer and communications security*, ser. CCS.
[43] K. Nissim, S. Raskhodnikova, and A. Smith, "Smooth sensitivity and sampling in private data analysis," in *Proceedings of the annual ACM symposium on Theory of Computing*, ser. STOC, 2007.
[44] M. Pettai and P. Laud, "Combining differential privacy and secure multiparty computation," in *Proceedings of the Annual Computer Security Applications Conference*, ser. ASAC, 2015.
[45] C. P. Pfleeger and S. L. Pfleeger, *Security in computing*, 2002.
[46] P. Pullonen, D. Bogdanov, and T. Schneider, "The design and implementation of a two-party protocol suite for sharemind 3," *CYBERNETICA Institute of Information Security, Tech. Rep.*, 2012.
[47] M. Rabin, "How to exchange secrets by oblivious transfer," *Technical Memo TR-81*, 1981.
[48] V. Rastogi and S. 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*, ser. SIGMOD, 2010.
[49] A. Shelat and M. Venkitasubramaniam, "Secure computation from millionaire," in *International Conference on the Theory and Application of Cryptology and Information Security*, ser. ASIACRYPT, 2015.
[50] A. Smith, A. Thakurta, and J. Upadhyay, "Is interaction necessary for distributed private learning?" in *IEEE Symposium on Security and Privacy*, 2017.
[51] G. Sood, "California Public Salaries Data," 2018. [Online]. Available: https://doi.org/10.7910/DVN/KA3TS8
[52] H. Takabi, S. Koppikar, and S. T. Zargar, "Differentially private distributed data analysis," in *IEEE International Conference on Collaboration and Internet Computing*, ser. CIC, 2016.
[53] A. D. P. Team, "Learning with privacy at scale," 2017. [Online]. Available: https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html
[54] M. L. G. ULB. (2018) Credit card fraud detection. [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud/data
[55] WWDC 2016, "Engineering privacy for your users," 2016. [Online]. Available: https://developer.apple.com/videos/play/wwdc2016/709/
[56] A. C.-C. Yao, "How to generate and exchange secrets," in *Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS, 1986.

APPENDIX

A. Garbled Circuit

Bellare et al. [3] formalize a *garbling scheme* as the tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, where Gb is probabilistic and all others are deterministic. A *string* is defined as a sequence of bits of finite length.

- $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f)$: Takes as input a security parameter $\lambda \in \mathbb{N}$ and the string f describing the original function to evaluate, $\mathsf{ev}(f, \cdot)$, and outputs string F describing the *garbled function*, $\mathsf{Ev}(F, \cdot)$, string e describing an *encoding function*, $\mathsf{En}(e, \cdot)$, and string d describing a *decoding function*, $\mathsf{De}(d, \cdot)$, as defined in the following.
- $X \leftarrow \mathsf{En}(e, x)$ is an *encoding function*, described by string e, that maps an *initial input* $x \in \{0,1\}^n$ to a *garbled input* X.
- $y \leftarrow \mathsf{De}(d, Y)$ is a *decoding function*, described by string d, that maps a *garbled output* Y to a *final output* y.
- $Y \leftarrow \mathsf{Ev}(F, X)$ is an *evaluation function*, described by string F, that maps a garbled input X to a *garbled output* Y.
- $y \leftarrow \mathsf{ev}(f, x)$ is an *evaluation function*, described by string f, that maps the input x to the output y, where $\mathsf{ev}(f, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^m$ is the *original function* we want to garble, and $n = f.n, m = f.m$ depend on f and must be computable from it in linear-time.

The following requirements are imposed on a garbling scheme:

- *Length condition*: If $f.n = f'.n, f.m = f'.m, |f| = |f'|$, $(F, e, d) \in [(\mathsf{Gb}(1^\lambda, f)]$, and $(F', e', d') \in [(\mathsf{Gb}(1^\lambda, f')]$, then $|F| = |F'|$, $|e| = |e'|$, and $|d| = |d'|$.
- *Non-degeneracy condition*: Let r are be the random coins of Gb. If $f.n = f'.n, f.m = f'.m, |f| = |f'|, (F, e, d) \in [(\mathsf{Gb}(1^\lambda, f; r)]$, and $(F', e', d') \in [(\mathsf{Gb}(1^\lambda, f'; r)]$, then $e = e'$ and $d = d'$.

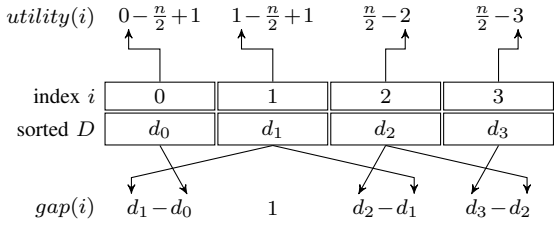


Fig. 12. *utility* and *gap* computed on sorted D with static access pattern.

- *Correctness condition*: If $f \in \{0,1\}^*, \lambda \in \mathbb{N}, x \in \{0,1\}^{f.n}$, and $(F,e,d) \in [\mathsf{Gb}(\lambda, f)]$, then $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e,x))) = \mathsf{ev}(f,x)$.

B. Static access pattern for utility and gap

Each party can compute *utility* (Definition 6) without any access to D, and *gap* (Definition 7) has a static access pattern, independent of the elements in (sorted) D, which makes the *gap* function *data-oblivious*, i.e., an attacker who sees the access pattern cannot learn anything about D. Fig. 12 visualizes how we compute *utility* and *gap* with static access pattern over sorted data D.

C. Non-decreasing Utility after Pruning

Theorem 5. *The input pruning from [1] does not decrease utility.*

Proof: Let $D_A = \{a_1, \ldots, a_m\}, D_B = \{b_1, \ldots, b_m\}$ with $a_1 < a_2 < \cdots < a_m$ and $b_1 < b_2 < \cdots < b_m$ (otherwise we use padding and uniqueness encoding from [1]). Let $a_i^s = D_A^s[i]$, i.e., the element at index i in the data of A after pruning step s. If some indices i, j, k exist such that $a_i^{s-1} < b_j^{s-1} \leq b_k^{s-1} < a_{i+1}^{s-1}$ where $b_j^{s-1}, \ldots, b_k^{s-1}$ are not in D_B^s but a_i^{s-1} is in D_A^s then pruning step s removed $b_j^{s-1}, \ldots, b_k^{s-1}$ but neither a_i^{s-1} nor a_{i+1}^{s-1}, one of which is further away from the median than $b_j^{s-1}, \ldots, b_k^{s-1}$. However, the utility of such a removed element either remains the same (it is a duplicate of a remaining element), or increases, i.e., they have the utility of their predecessor (resp., successor) in D^s. Since one of the elements a_i^{s-1}, a_{i+1}^{s-1} is closer to the median after pruning step s than before, its utility increases and so does the utility for all elements between a_i^{s-1} and a_{i+1}^{s-1}.

If no such indices i, j, k exist, then we only remove the elements furthest away from the median and the utility for remaining elements is unchanged. The utility for removed element x either remains the same (x is equal to a remaining element) or increases. The latter is due to the fact that removed elements have the same rank-based distance to the median, either $\text{rank}_{D^s}(x) = 0$ or $\text{rank}_{D^s}(x) = |D^s|$. Since $|D^s| < |D^{s-1}|$ we have $u_{\text{med}}(D^s, x) > u_{\text{med}}(D^{s-1}, x)$. ∎

An example of non-decreasing utility after pruning is shown in Table III for unique elements. For example, element a_1 has utility -3 before pruning, after pruning its utility increases to -2, whereas the utility for b_2, a_3 remain as before.

TABLE III. UTILITY DOES NOT DECREASE FOR SORTED $D = D_A \cup D_B$ BEFORE AND AFTER ONE PRUNING STEP WITH $D_A = \{a_1, \ldots, a_4\}$, $D_B = \{b_1, \ldots, b_4\}$.

D	a_1	b_1	a_2	b_2	a_3	b_3	a_4	b_4
$u_{\text{med}}(D, \cdot)$	-3	-2	-1	0	0	-1	-2	-3
D^1	–	b_1	–	b_2	a_3	–	a_4	–
$u_{\text{med}}(D^1, \cdot)$	-2	-1	-1	0	0	-1	-1	-2

Algorithm 5 Algorithm PAD pads the input of party $P \in \{A, B\}$ such that the element with rank k is at the median position (part of FIND-RANKED-ELEMENT from [1]).

Input: Data D_P, rank k, padding $\widehat{p}$
Output: Input padded to place k^{th}-ranked element at median position of the union of D_A, D_B

1: Sort D_P and retain only the k smallest values
2: Pad D_P with $+\infty$ until $|D_P| = k$
3: Pad D_P with $\widehat{p}$ until $|D_P| = 2^{\lceil \log_2(k) \rceil}$
4: **return** D_P

D. Padding

In a preprocessing step to the actual pruning the data is padded as described in Algorithm 5 where A calls $\text{PAD}(D_A, k, +\infty)$ and B calls $\text{PAD}(D_A, k, -\infty)$ with $k = \lceil (|D_A| + |D_B|)/2 \rceil$. Note that the data size of each party, i.e., $|D_A|, |D_B|$, can be hidden via additional padding.

E. Merge Implementation

For the merging implementation, as seen in Algorithm 6, we use the bitonic mergers as described in [31, Section 5.1] which require a bitonic list as input, i.e., a list that is monotonically increasing then decreasing (or vice versa). Bitonic merging recursively splits the list in halves and compares and swaps elements such that every element of one half is greater than every element of the other half.

F. Random Draw

We implemented RANDOMDRAW, see Algorithm 7, with rejection sampling using efficient operations, namely XOR, OR, AND, comparison. Rejection sampling is unbiased, however, for a fixed input size of k nonces it might abort with probability at most 2^{-k}[11]. Rejection sampling (without abort) is used in Apple's macOS [37]. For our evaluation in Section VI we used $k = 20$.

An alternative to rejection sampling is a slightly biased sampling algorithm without abort requiring only one nonce instead of k per party: If the masked XOR of nonces (r) is larger than M one uses $r - M$ as the sampled output. We compared biased sampling with rejection sampling ($k = 20$) using the median of 20 runs for our largest circuit ($\epsilon = 0.25, |D| = 2 \cdot 10^6$) with approximately 100 ms delay and

[11] We now consider the worst-case rejection rate, i.e., comparison $r < M$ in line 11 of Algorithm 7. Recall that r is the XOR of uniform random values, thus, each bit in r is uniform random as well. Masking ensures that at most the $mask$ first bits of r are set, in effect reducing the size of r to $mask$. The number of set bits (i.e., bits with value 1) in M influences the rejection probability. The rejection rate is maximized if only one bit in M is set. Then, r is rejected with probability $1/2$ as all r with 0 at position $mask$ are accepted ($r < M$), while the other half is rejected. Increasing the number of set bits in M decreases the rejection rate (as more r can be smaller than M). Thus, the rejection probability per sample r is at most $1/2$.

Algorithm 6 Algorithm MERGE returns sorted $D^s = D^s_A \cup D^s_B$.

Input: Left index l, right index r, bitonic list D^s.
Output: Sorted D^s.

return if $r < l$
$m \leftarrow l + \frac{r-l}{2}$
for $i \leftarrow l$ **to** m **do**
  $e \leftarrow i + \left\lfloor \frac{r-l}{2} + 1 \right\rfloor$
  Swap d^s_i with d^s_e **if** $d^s_i > d^s_e$
end for
MERGE$(l, m-1, D^s)$
MERGE$(m+1, r, D^s)$

Algorithm 7 Algorithm RANDOMDRAW with parties A, B based on [37].

Input: Max. value M, lists of k nonces N_A, N_B from A, B.
Output: Uniform random integer in $[0, M)$

```
//Find most significant 1-bit in M, set following
  bits to 1 in mask
```
$c \leftarrow 0$
$mask \leftarrow 0$
for $i \leftarrow$ bitlength b **to** 1 **do**
  $c \leftarrow c$ OR i^{th} bit of M
  i^{th} bit of $mask \leftarrow c$
end for

```
//Rejection sampling with abort
```
$s \leftarrow \perp$
for $i \leftarrow 1$ **to** k **do**
  $r \leftarrow N_A[i]$ XOR $N_B[i]$
  $r \leftarrow r$ AND $mask$
  if $r < M$ **then**
    $s \leftarrow r$
  end if
end for
if $s = \perp$ **then**
  abort
end if
return r

100 MBits/s bandwidth. Biased sampling required around 28k fewer gates and sent 400 KB less than rejection sampling with $k = 20$, which corresponds to a reduction in circuit size and communication of less than 1% for GC and around 3–4% for GC + SS. The runtime with biased sampling decreased by 2.2 seconds for GC (18.5% faster) but only by 0.18 seconds for GC + SS (2.6%). (For $k = 30$ an additional 44k gates and 600 KB are required compared to biased sampling, leading to similar runtimes as for $k = 20$.) Thus, we use rejection sampling as it is unbiased with only small impact on the runtime of GC + SS.

G. Simulation Proof for Pruning

Aggarwal et al. [1] prove the security of their exact k^{th}-ranked element computation in the semi-honest model by showing that A (similarly B) can simulate the secure protocol given its own input D_A, and the value m of the k^{th}-ranked element. We reproduce their simulation in the following as we use the same argument with small modifications.

The simulation executed by A (similarly B) in [1] is detailed in Algorithm 8. If the data D_A contains duplicates, $\lceil \log_2 |D_A| \rceil + 1$ bits are added to the binary representation of each element to make it unique as required for the simulation. E.g., A adds for each element the bit 0 followed by the rank of the element in the least significant bit positions. B follows the same procedure using 1 instead of 0. These bits are removed from the final output.

Algorithm 8 Algorithm SIMULATEPRUNING simulates the secure k^{th}-ranked element computation from [1].

Input: Parameter element rank k, real execution result m and iteration count j. Note that D_A is known to A and all items in $D_A \cup D_B$ are distinct.
Output: Simulation of running the protocol for finding the k^{th}-ranked element m in $D_A \cup D_B$.

A initializes $D^1_A \leftarrow$ PAD$(D_A, k, +\infty)$ `//Appendix D`
for $i \leftarrow 0$ **to** $j - 1$ **do**
  A computes $m_A \leftarrow$ median of D^i_A
  Secure comparison result c is set to 1 if $m_A < m$ (i.e., $m_A < m_B$) otherwise it is 0
  A sets $D^{i+1}_A \leftarrow$ upper half of D^i_A if $c = 1$ otherwise it is the lower half
end for
The final secure comparison result c is set to 1 if $m_A < m$ and else it is 0

Aggarwal et al. [1] execute the simulation as SIMULATEPRUNING$(k, m, \lceil \log_2(k) \rceil)$, i.e., full pruning until only one element remains. Lemma 2 from [1] states that the transcript of the real execution and the simulated execution are equivalent. Additionally, the state information, i.e., pruned data D^i_A, that A has at each iteration i is the same as well.

In our protocol we do not perform the full execution, i.e., only s iterations. We do not know the exact value m, however, A knows its state D^s_A at the final step and we use median of D^s_A instead of m. Altogether, we call the simulation with SIMULATEPRUNING$(k,$ median of $D^s_A, s)$.

We now show by contradiction that our simulation outputs the correct comparison results. Assume $c = 1$, i.e., $m_A < m_B$, at iteration i in our real execution but our simulation outputs 0, i.e., $m_A \geq$ median of D^s_A. Then D^{i+1}_A is the lower half of D^i_A and only elements smaller than or equal to $m_A =$ median of D^i_A remain in D^{i+1}_A and thus in D^s_A. In other words, for $x \in D^{i+1}_A$ we have $x \leq m_A$ and due to $D^s_A \subseteq D^{i+1}_A$ we have median of $D^s_A < m_A$. However, this contradicts $m_A \geq$ median of D^s_A, i.e., output 0. Analogously, we find a contradiction if $c = 0$ in our real execution but 1 in the simulation.

H. Further PRUNE-neighboring evaluation

Table IV shows the average number of changes in a data set D_B to create a neighbor that is not a PRUNE-neighbor w.r.t. D_A. The number of changes corresponds to the average group privacy we can expect. Each additional pruning step increases the possibility to find a non-PRUNE-neighbor. Thus, we use $\epsilon = 2$ as it leads to the most number of pruning steps in our evaluation.

Table V shows that lower values of ϵ provide higher group privacy. We list the detailed minimum and average number of

TABLE IV. **AVERAGE CHANGES** IN D_B TO SAMPLE A NEIGHBOR THAT IS NOT A PRUNE-NEIGHBOR W.R.T. D_A. EVALUATED FOR 52 000 NEIGHBORS (ALL COMBINATIONS OF UP TO 50 REMOVALS AND 50 ADDITIONS WITH 20 SAMPLES PER COMBINATION) EACH ROW SHOWS THE AVERAGE CHANGES FOR $\epsilon = 2$ WITH 95% CONFIDENCE INTERVAL AND >100 INDICATES NONE WERE FOUND FOR UP TO 100 CHANGES.

D_A \ D_B	*Wages [51]*	*Transactions [54]*	*Times [54]*	*Payments [11]*	*Weights [33]*	*Quantities [33]*
Wages [51]	58.6 ± 0.26	50.7 ± 0.25	49.7 ± 0.13	50.0 ± 0.17	53.9 ± 0.26	50.9 ± 0.24
Transactions [54]	76.6 ± 9.59	50 ± 0.18	50.5 ± 0.26	48.5 ± 0.18	72.3 ± 0.52	55.6 ± 0.16
Times [54]	63.7 ± 0.22	64.9 ± 0.20	50.3 ± 0.18	50 ± 0.25	61.2 ± 0.20	62.5 ± 0.10
Payments [11]	68.9 ± 0.35	59.8 ± 0.19	>100	50 ± 0.15	71.4 ± 1.26	57.9 ± 0.13
Weights [33]	55.0 ± 1.77	49.6 ± 0.15	50.9 ± 0.18	50.7 ± 0.14	61.2 ± 0.20	50.5 ± 0.24
Quantities [33]	68.3 ± 0.63	64.7 ± 0.31	51 ± 0.25	51 ± 0.25	54.5 ± 0.18	59.6 ± 0.13

TABLE V. **AVERAGE & MINIMUM CHANGES** IN D_B TO SAMPLE A NEIGHBOR THAT IS NOT A PRUNE-NEIGHBOR W.R.T. D_A, WHERE D_A CONSITS OF 284K CREDIT CARD TRANSACTIONS [54]. EVALUATED FOR 52 000 NEIGHBORS (ALL COMBINATIONS OF UP TO 50 REMOVALS AND 50 ADDITIONS WITH 20 SAMPLES PER COMBINATION) EVALUATED FOR $\epsilon \in \{0.25, 0.5, 1, 2\}$ (WITH 95% CONFIDENCE INTERVAL FOR AVERAGE), AND >100 INDICATES NO VIOLATION WAS FOUND FOR UP TO 100 CHANGES.

D_B	ϵ	*Avg.*	*Min.*
Open Payments [11] (6M payments)	0.25	>100	>100
	0.5	>100	>100
	1	50.1 ± 0.26	37
	2	48.5 ± 0.18	30
California public salaries [51] (71k wages)	0.25	>100	>100
	0.5	>100	>100
	1	76.6 ± 25.38	65
	2	76.6 ± 9.59	65
Walmart supply chain [33] (175k shipment weights)	0.25	>100	>100
	0.5	>100	>100
	1	72.3 ± 0.73	36
	2	72.3 ± 0.52	36
Walmart supply chain [33] (175k shipment quantities)	0.25	>100	>100
	0.5	>100	>100
	1	55.6 ± 0.23	23
	2	55.6 ± 0.16	23
Credit card [54] (284k transaction times)	0.25	>100	>100
	0.5	>100	>100
	1	>100	>100
	2	50.5 ± 0.26	20

changes for $\epsilon \in \{0.25, 0.5, 1, 2\}$ where D_A consists of credit card transactions from [54]. Note that we list the minimum over all pruning steps (i.e., the value for minimum changes can be the same for different pruning steps and their corresponding epsilon value).