Scaling Two-Party Differentially Private Selection

by

Haoyan Ni

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Haoyan Ni 2026

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We consider the problem of differentially private (DP) selection in the two-party setting. This problem can be solved with excellent utility guarantee in the central setting, but the distributed case is much less studied. Existing solutions use secure multi-party computation (MPC) techniques to simulate computation in the central model, which are not sufficiently scalable to large candidate sets.

This work provides a new protocol for two-party DP selection that achieves sublinear runtime in the MPC phase. Our design lets each party locally trim the candidate set before participating in an MPC protocol. Based on this heuristic, we provide two variations, one of which reveals each party's trimmed candidate set and the other does not. We evaluate our method on public datasets based on review counts and location check-ins. The results demonstrate that the variant hiding the trimmed candidate sets outperforms the other variant in both utility and efficiency. Furthermore, our solution is able to offer competitive utility to the traditional solution at a significantly lower computation cost in lower privacy regimes.

Acknowledgements

I want to thank my supervisor, Florian Kerschbaum. His boundless patience and kindness have been a great encouragement for me in the past two years. I would also like to thank everyone in the CrySP lab for creating a welcoming and supportive environment. Lastly, thanks to my parents for enabling me to study at Waterloo in the first place, and for their unconditional emotional support in my difficult times.

Dedication

This is dedicated to my grandfather.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

Dedication v

List of Tables viii

1 Introduction 1

2 Preliminary 3

2.1 Differential Privacy 3

2.2 Garbled Circuit 5

2.3 Problem Statement 5

3 Related Work 6

3.1 DP Selection in the Central Model 6

3.2 Distributed DP 7

3.3 DP Selection in the Distributed Model 7

4 Baseline Method **9**
4.1 Algorithm in the Central Model . . . 9
4.2 Secure Computation of DP Selection . . . 11
4.2.1 Oblivious Swapper . . . 12
4.2.2 Oblivious Top-k Selection . . . 12
4.2.3 Shuffling . . . 14
4.2.4 Complete Protocol . . . 14

5 Threshold Trimming Algorithm **16**
5.1 Heuristic in the Non-private Setting . . . 16
5.2 Incorporating DP . . . 17
5.2.1 Proof of Privacy . . . 18
5.3 MPC Implementation . . . 26
5.3.1 Constructing the Sum Array . . . 27

6 Joint EM Trimming Algorithm **30**
6.1 Relaxing the Selection Problem . . . 30
6.2 Applying Exponential Mechanism . . . 31
6.3 Efficient Sampling . . . 32
6.4 Complete Protocol . . . 35

7 Experimental results **37**
7.1 Datasets . . . 37
7.2 Hyperparameter Selection . . . 37
7.3 Utility . . . 39
7.4 Efficiency . . . 41
7.5 Effects of Hyperparameters . . . 42

8 Conclusion **44**

References **45**

List of Tables

5.1 Variable definitions in the proof of Theorem 5 22

7.1 Dataset details 38
7.2 Utility results 40
7.3 Standard deviation results 41
7.4 Gate counts 42
7.5 Effect of m on utility of the threshold trimming algorithm 43
7.6 Effect of s on utility of the joint EM trimming algorithm 43

Chapter 1

Introduction

Given a finite set of items where each item has an associated utility value, the selection problem aims to determine the top-k items with the highest utilities. Selection is one of the most fundamental problems in data science, as it abstracts a variety of data analysis tasks such as feature selection and empirical risk minimization. When the utility function is data-sensitive, the selection result may reveal private information of dataset participants. Hence, special care must be taken to ensure that the selection algorithm does not compromise any individual's privacy in the dataset.

Differential privacy (DP) offers a promising solution by ensuring that the output of the algorithm is random and remains insensitive to any single data point in probability distribution [15]. Although differentially private selection has been extensively studied before [15, 25, 3, 16, 28, 3], most proposed solutions operate in the central model, which requires a single data-analyzing entity who is trusted by all dataset participants. This setting is often unrealistic. An alternative model, local differential privacy, weakens this trust assumption by requiring every data point to be made private before leaving the data subject [15]. However, solutions adopting local differential privacy typically require a massive, often impractical, number of users to achieve accurate results.

Distributed differential privacy offers a middle ground between the two privacy models, where the dataset is shared between multiple parties [14]. This model relaxes the trust assumption in the central model, as each data subject only needs to trust one data-holding party out of many. Existing solutions to differentially private selection in the distributed model use secure multi-party computation (MPC) techniques to simulate computation in the central model [5, 10]. These solutions have linear time complexity in the MPC phase which leads to significant computation cost with large candidate set.

We focus on the two-party case and seek to develop data-dependent algorithm for differentially private top-k selection that achieves sublinear runtime in the MPC phase. Our algorithm is based on the heuristic that in the selection process, a candidate with outstanding combined utility on both parties' datasets should also have high utility on at least one party's dataset. Hence, we let each party locally trim the candidate set and then let both parties collectively perform an MPC selection procedure on the two trimmed candidate sets. [1] Following this heuristic, we develop two ideal functionalities that solve the DP selection problem. We then describe how to realize these two functionalities using MPC techniques. Finally, we conduct an empirical evaluation of our solutions against naive solutions in public datasets.

The remainder of this thesis is organized as follows. In Chapter 2, we introduce differential privacy and the problem definition. In Chapter 3, we review the relevant literature. Chapter 4 describes a naive baseline protocol. Chapter 5 and 6 describe our main algorithms and prove their privacy guarantees. Finally, we empirically evaluate our algorithms in Chapter 7.

[1] Note that the algorithm may return biased results even in the absence of privacy constraints.

Chapter 2

Preliminary

2.1 Differential Privacy

Differential privacy is a mathematical framework that introduces randomness in dataset queries to provide provable privacy guarantees for individual data points in the dataset [15]. Over recent years, DP has become increasingly popular among both privacy theorists and practitioners. A particularly noteworthy example is the adoption of DP in the 2020 U.S. Census [24]. At a high level, an algorithm that satisfies DP produces nearly statistically indistinguishable output distributions with or without the inclusion of any single data point. The formal definition is as follows.

Definition 1. *(Differential privacy) [15] Let $\epsilon > 0, \delta \geq 0$. A randomized algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ satisfies (ϵ, δ)-DP, if for all pairs of neighboring datasets $D, D' \in \mathcal{D}$ and all $T \subseteq \mathcal{R}$, we have:*

$$\mathbb{P}(\mathcal{M}(D) \in T) \leq e^{\epsilon}\mathbb{P}(\mathcal{M}(D') \in T) + \delta.$$

The above definition does not specify what *neighboring* means, as it depends on the problem context. In many typical studies, as in the rest of this thesis, we say that two datasets D, D' are neighboring if one can be obtained from the other with the inclusion/exclusion of a single data point. We denote this relationship by $D \sim D'$. The resulting definition is commonly referred to as unbounded differential privacy.

The parameter ϵ in Definition 1, often referred to as the privacy budget, captures how close the outputs $\mathcal{M}(D)$ and $\mathcal{M}(D')$ are in probability distribution. The smaller ϵ is, the

more similar are $\mathcal{M}(D)$ and $\mathcal{M}(D')$, and therefore the better privacy $\mathcal{M}$ provides. The parameter δ provides a slight relaxation to the definition, by allowing the privacy guarantee to fail in some small probability event. When $\delta = 0$, the algorithm $\mathcal{M}$ is also said to satisfy *pure* DP. In this case, we can drop δ in the notation and simply say that $\mathcal{M}$ satisfies ϵ-DP. In the opposite case when $\delta > 0$, we say that $\mathcal{M}$ provides *approximate* DP.

The definition of DP has many useful mathematical properties which makes it amenable for analysis. We list two such properties below.

Lemma 1. *(Post-processing)* [15] *If $\mathcal{M} : \mathcal{D} \to R$ is ϵ-DP, then $A \circ \mathcal{M}$ is also ϵ-DP for any function A applied on the output of $\mathcal{M}$.*

Lemma 2. *(Basic composition)* [15] *If $\mathcal{M}_1 : \mathcal{D} \to R_1$ is (ϵ_1, δ_1)-DP and $\mathcal{M}_2 : R_1 \times \mathcal{D} \to R_2$ is (ϵ_2, δ_2)-DP, then the map $[D \mapsto \mathcal{M}_2(\mathcal{M}_1(D), D)]$ is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DP.*

The privacy guarantee of DP is information-theoretic in the sense that it does not concern an adversary's computation power. In two-party setting where informational-theoretic security is infeasible to achieve, it is common to opt for computational DP, which is captured by the following definition.

Definition 2. *(IND-CDP-2PC) [22] A two-party protocol Π between parties A, B provides ϵ-IND-CDP for B, if for any probabilistic polynomial adversary T and any pair of neighboring datasets $D_B \sim D'_B$, the view $VIEW_A^{\Pi}(D_A, \cdot)$ of A satisfies*

$$\mathbb{P}(T(VIEW_A^{\Pi}(D_A, D_B) = 1) \leq e^{\epsilon}\mathbb{P}(T(VIEW_A^{\Pi}(D_A, D'_B) = 1) + \delta_\kappa,$$

where δ_κ is a negligible term associated with a security parameter κ.

In essence, this definition says that any information that party A can derive from the protocol in polynomial time must be DP with respect to party B's dataset.

We conclude this section with the following example of an actual DP mechanism. Consider the objective of finding an item in a candidate set that maximizes a given utility function, with respect to some private dataset. The *exponential mechanism* provides a method to do so while conforming to the DP guarantee.

Definition 3. *(Exponential mechanism)* [15] *Let utility function $u : \mathcal{D} \times O \to \mathbb{R}$ be given, and let $\Delta = \max_{D, D' \sim \mathcal{D}, o \in O} |u(D, o) - u(D', o)|$. The exponential mechanism $\mathcal{M}$ samples from O with probability distribution*

$$\mathbb{P}(\mathcal{M}(D) = o) \propto \exp(\frac{\epsilon \cdot u(D, o)}{2\Delta}),$$

where $\propto$ elides the normalization factor. Δ is also called the sensitivity of u.

Theorem 1. [15] *The exponential mechanism is ϵ-DP.*

The exponential mechanism is remarkably flexible and can be applied to a variety of data analysis problems. However, it may be prohibitively time-consuming when the candidate set O is very large or possibly infinite.

2.2 Garbled Circuit

Yao's garbled circuit protocol is a general procedure for two parties to securely evaluate a static circuit. The details of the construction are unimportant for our purpose, as we are only concerned with the underlying circuit that would be computed. With modern optimization techniques for garbled circuit, the computation cost and communication size of the protocol can be considered directly proportional to the number of AND gates in the evaluated circuit [23]. Garbled circuits are secure against semi-honest adversaries, but can also be made secure against malicious adversaries using additional techniques, albeit at the cost of increasing computation overhead [17].

2.3 Problem Statement

We consider the problem of identifying an *unordered* index set of the k largest entries in a count vector $x \in \mathbb{N}^d$. This vector x is the sum of some user-associated vectors in $\{0,1\}^d$, each of which is a data point that need to be protected by DP. Since the user-associated vectors can be arbitrary, we may say that two count vectors x, x' are neighboring if $x - x' \in \{0,1\}^d$ or $x' - x \in \{0,1\}^d$. This problem definition captures many settings. One example is choosing k items with most reviews in an online store, where every user may give at most one review to any item.

For our study, we assume that the count vector to be selected from is split by two parties. More precisely, let party P_1, P_2 each hold a private count vector $x_1, x_2 \in \mathbb{N}^d$. Our goal is to design a protocol that computes an unordered set of k approximately largest entries in $x_1 + x_2$, while providing computational DP for both parties.

Chapter 3

Related Work

3.1 DP Selection in the Central Model

For selecting the index of a single largest entry, the report noisy max mechanism introduced by Dwork and Roth is a well known solution [15]. This algorithm simply adds Laplace noise to each entry before selecting the largest noisy entry. The exponential mechanism is also a common alternative, which offers competitive practical performance [25]. More recently, McKenna and Sheldon proposed the permute-and-flip mechanism, which Pareto dominates the exponential mechanism [25]. Interestingly, both the exponential mechanism and permute-and-flip have been shown to be equivalent to the report noisy max mechanism with Gumbel and exponential noise respectively [13, 11]. Report noisy max can also be used with geometric noise to provide DP [5]. [1] Finally, regardless of the noise distribution used (Laplace, Gumbel, exponential, geometric), all versions of report noisy max provide asymptotically equivalent utility-privacy tradeoff.

The more generalized problem of top-k selection is also extensively studied. A common solution is to repeatedly apply the exponential mechanism to select the largest entry in the unselected ones. This solution is known as the peeling technique. With the use of DP composition theorem, the peeling technique has been shown to be asymptotically optimal in the approximate DP setting [3]. Despite this result, the JOINT mechanism proposed by Gillenwater et al. [16] is reported to achieve superior empirical utility in practice. Their work devises an efficient sampling method which directly realizes the exponential mechanism on the set of all length-k sequences in the entire index domain. In parallel to

[1] Note that with geometric noise, there will need to be a tie-breaking procedure.

this line of research, two other works [28, 35] have explored data-dependent algorithms for top-k selection. Both of these works aim to exploit large gaps between adjacent entries in a sorted vector. The overall idea is that if the difference between the k-th and $(k+1)$-th largest entries is probably large, then one can directly return the unordered top-k index set, as a neighboring count vector could only change the top-k index set with small probability. These algorithms could offer better utility-privacy tradeoffs in certain use cases. In particular, Zhu and Wang demonstrated that their data-dependent selection algorithm significantly improves the performance of a "Private Aggregation of Teacher Ensembles (PATE)" model in multi-label classification tasks [35].

3.2 Distributed DP

Motivated by settings where there does not exist a single entrusted data curator, other DP models have emerged. The local DP model offers the strongest privacy guarantee by ensuring that each data point is protected by DP at the point of its release. However, achieving local DP requires perturbing every data point in the dataset, which may result in unacceptable utility loss. This predicament has inspired subsequent works to focus on models that lie in between the central model and the local model [6]. The distributed model is one of them, which our work aims to contribute. In short, the distributed model assumes that the private dataset is split across multiple servers, and the servers communicate between each other to produce a public output. A recurring theme in distributed DP protocols is to use MPC techniques to simulate computation in the central model.

3.3 DP Selection in the Distributed Model

The first secure implementation of report noisy max in a distributed setting was given by Champion et al. [5]. Their implementation uses an oblivious stack to sample independent Bernoulli variables, which can then be converted into geometric noises used in report noisy max. The importance of this work lies in that the computation cost grows only poly-logarithmically to the security parameter. Despite this efficiency improvement, the computation cost for using oblivious stack may still be too high for practical application. Steinke suggested a more practical solution, which uses distributed noise generation and secret sharing scheme to simulate report noisy max in the central model [29]. Distributed noise generation lets each party locally sample a partial noise, and then sum them together

in the MPC phase. This approach makes the protocol much more efficient, although it introduces extra noise in the selection procedure and therefore results in a bigger utility loss. A subsequent work by Damgård et al. [10] implemented this solution with some slight modifications, thus affirming its feasibility. Another related work by Choi et al. [8] explored the possibility of instantiating exponential mechanism in the two-party setting, as an application of the product sampling problem. Their work makes use of fully homomorphic encryption in the sampling procedure, and is likely of theoretical interest only due to its heavy computation overhead.

Chapter 4

Baseline Method

To tackle the problem of two-party DP selection, we start by deriving a baseline method from the famous report noisy max mechanism. We first show that report noisy max with geometric noise can be generalized to top-k selection for any $k \geq 1$. After that, we provide an implementation of this mechanism in a two-party secure computation setting. This baseline method provides the building blocks for our next studies.

4.1 Algorithm in the Central Model

We start by recalling the report noisy max mechanism in the central setting. The algorithm adds Laplace noise to each vector entry and then reveals the index that maximizes the noisy entry. The report noisy max mechanism can be shown to work with several other noise distributions, all with asymptotically equivalent private-utility tradeoff [13, 11, 5]. We use geometric noise, which has non-negative integers as its support. This choice is natural, as integer-based computation is more easily supported in the MPC setting. In addition, DP mechanisms that use floating-point computation must take special care to avoid floating point vulnerabilities [27], which could introduce significant extra complexity in the MPC implementation.

To efficiently generalize the solution above for top-k selection where $k \geq 1$, we propose the oneshot top-k report noisy max mechanism. In this algorithm, we add geometric noise to each vector entry as before and then report the *unordered* index set of the k largest noisy entries. Note that selecting k indices scales the privacy budget by k. The DP proof of this algorithm is likely folklore, but we include it below for completeness.

Algorithm 1 Oneshot top-k report noisy max with geometric noise

Input: $x \in \mathbb{N}^d, k \geq 1, \epsilon > 0$

1: sample $\eta \sim \text{Geometric}(1 - e^{-\epsilon})^d$

2: $\tilde{x} = x + \eta$

3: **return** top-k index set of $\tilde{x}$ (break ties by preferring smaller indices)

Theorem 2. *Algorithm 1 provides $k\epsilon$-DP.*

Proof. Let Algorithm 1 be denoted by $\mathcal{M}(\cdot)$, x, x' be two neighboring count vectors and o be a possible output.

Let $\mathcal{G}$ be the set of all random vectors η such that $\mathcal{M}(x)$ outputs o. Define $\mathcal{G}'$ for x' similarly. We claim that for all $a \in \mathcal{G}$, we have: $a + \mathbb{1}_o \in \mathcal{G}'$, where $\mathbb{1}_o$ is the vector with 1 in every index in o, and 0 everywhere else. This claim can be verified by showing that for all $a \in \mathcal{G}, i \in o$ and $j \notin o$, i still precedes j when ranked by entries in $x' + a + \mathbb{1}_o$.

Fix $a \in \mathcal{G}, i \in o, j \notin o$. Since i is in the top-k index set of $x + a$ but j is not, either (a): $(x + a)[i] > (x + a)[j]$ or (b): $i < j$ and $(x + a)[i] = (x + a)[j]$. Furthermore,

$$(x' + a + \mathbb{1}_o)[i] - (x' + a + \mathbb{1}_o)[j] = (x' + a)[i] - (x' + a)[j] + 1 \geq (x + a)[i] - (x + a)[j].$$

As a result, either $(x'+a+\mathbb{1}_o)[i] > (x'+a+\mathbb{1}_o)[j]$, or $i < j$ and $(x'+a+\mathbb{1}_o)[i] = (x'+a+\mathbb{1}_o)[j]$. Since the choice of i, j is arbitrary over o and $\{0, \ldots, d-1\} \setminus o$, this conclusion shows that o is still the top-k index set in $x' + a + \mathbb{1}_o$, which proves $a + \mathbb{1}_o \in \mathcal{G}'$.

Now, we have:

$$\begin{aligned}
&\mathbb{P}(\mathcal{M}(x) = o) \\
=&\sum_{a \in \mathcal{G}} \mathbb{P}(\eta = a) \\
=&e^{k\epsilon} \sum_{a \in \mathcal{G}} \mathbb{P}(\eta = a + \mathbb{1}_o) \\
\leq&e^{k\epsilon} \sum_{a \in \mathcal{G}'} \mathbb{P}(\eta = a) \qquad (\mathcal{G} + \mathbb{1}_o \subseteq \mathcal{G}') \\
=&e^{k\epsilon} \mathbb{P}(\mathcal{M}(x') = o),
\end{aligned}$$

where the second equality holds because increasing each entry of a by 1 scales the probability mass $\mathbb{P}(\eta = a)$ down by $e^{-\epsilon}$. This concludes our proof. □

An alternative solution to the top-k selection problem is to apply report noisy max k times, each time selecting the largest entry in the remaining unselected ones. This method is commonly denoted by the peeling technique in DP literature. One may perform a tighter analysis on its privacy budget in the approximate DP setting by utilizing DP composition theorem. To limit the scope of our study, we leave this possibility to future work. We also highlight that implementing the peeling technique in an MPC setting would come at the cost of a higher round complexity which we would like to avoid.

4.2 Secure Computation of DP Selection

Before we provide a secure implementation of Algorithm 1 in a multi-party setting, we first describe the ideal functionality that we want to achieve, which is detailed in Algorithm 2. This functionality is mostly similar to Algorithm 1, except that both parties add geometric noise to their private vectors and then sum them together, which results in a larger amount of noise added to the sum of two private vectors. Note that it is insufficient for only one party to perturb their private vector, for in that case the algorithm would be a deterministic function parameterized by their input, thus violating DP for the other party. This type of approach where each party samples a partial noise and sums them together in an MPC protocol, is commonly titled distributed noise generation in DP literature. Distributed noise generation is highly efficient as it only requires performing ADD operation in the MPC phase. For two parties to jointly sample geometric noises, the state-of-the-art solution relies on using oblivious data structure [5], which is much slower in comparison.

Algorithm 2 Ideal functionality

Input: P_1, P_2 hold private vectors $x_1, x_2 \in \mathbb{N}^d$ respectively, $k \geq 1, \epsilon > 0$

1: P_1, P_2 each privately samples $\eta_1, \eta_2 \sim \text{Geometric}(1 - e^{-\epsilon})^d$, respectively
2: $\tilde{x}_1 = x_1 + \eta_1$
3: $\tilde{x}_2 = x_2 + \eta_2$
4: **return** top-k index set of $\tilde{x}_1 + \tilde{x}_2$ (break ties by preferring smaller indices)

It is straightforward to show that Algorithm 2 is $k\epsilon$-DP for both parties. In the view of P_1, the output of Algorithm 2 is equivalent to that output from calling Algorithm 1 with $\tilde{x}_1 + x_2$, where $\tilde{x}_1 + x_2 \sim \tilde{x}_1 + x_2'$ whenever $x_2 \sim x_2'$. Hence, the DP property of Algorithm 2 follows from Theorem 2.

Corollary 1. *Algorithm 2 provides $k\epsilon$-DP.*

We now explain how to construct a secure protocol that simulates Algorithm 2. In this protocol, both parties start by locally producing a noisy vector, and then securely compute their sum inside a garbled circuit. Next, they run an oblivious top-k selection network to compute the top-k indices of the noisy sum vector (Section 4.2.2). The indices cannot be released at this stage however, as their ordering may violate DP. Hence, the parties must perform an oblivious shuffle on this index set to remove the ordering information before releasing them (Section 4.2.3).

4.2.1 Oblivious Swapper

The oblivious swapper is a basic primitive in garbled circuit. An oblivious swapper takes two inputs x, y and a control bit s, and then swap x and y exactly when s is 1. The typical implementation of oblivious swapper computes the XOR value of x and y and then conditionally set it to 0 when s is 0. This final value is then XORed with each of x and y to produce the final output.

4.2.2 Oblivious Top-k Selection

After the circuit adds two noisy vectors, the next step is to select the top-k indices in the sum. The challenge of this operation is that the selection process must be done with a fixed sequence of comparators. A trivial solution is to apply the tournament method k times, where each application loops over the vector and finds the remaining largest entry. This naive solution uses $O(kd)$ comparators. To reduce the number of comparators, we adopt Yao's top-k selection network [33] in our implementation, which we recall below.

Yao's selection network starts by partitioning the d input entries into two halves $(u[0], u[1], \ldots, u[\lceil d/2 \rceil - 1]), (v[0], v[1], \ldots, v[\lfloor d/2 \rfloor - 1])$ such that $u[i] \geq v[i]$ for all $0 \leq i < \lfloor d/2 \rfloor$. Observe that the final top-k entries contains at most $\lfloor k \rfloor / 2$ elements from v. The selection network thus uses recursion to find the top-$(\lfloor k \rfloor / 2)$ entries from v. It then combines this output with u and recurses on the result. This procedure reduces the number of input entries by approximately $(d - k)/2$, while leaving k unchanged.

The above recursion process terminates in one of two cases: $k \leq 2$ or $k > d/2$. In the former case, we apply the naive tournament method to find the top-k entries. In the latter case, the algorithm exchanges the roles of k and $d - k$, by finding the smallest $d - k$ entries and returning the remaining ones. We provide a pseudocode of Yao's top-k selection network in Algorithm 3.

Algorithm 3 Yao's oblivious top-k network

Input: Array x of size d and $1 \leq k \leq d$

Output: Move the top-k entries of x to the front

```
function YAO(x, k)
    if k ≤ 2 then
        use the tournament method to move the top k entries of x to the front
    else if k > d/2 then
        use recursion to move the bottom d − k entries of x to the back
    else
        for i = 0, ..., ⌊d/2⌋ − 1 do
            swap x[i] and x[⌈d/2⌉ + i] if x[⌈d/2⌉ + i] > x[i]
        Yao(x[⌈d/2⌉ ... d − 1], ⌊k/2⌋)
        Yao(x[0 ... ⌈d/2⌉ + ⌊k/2⌋ − 1], k)
```

Theorem 3. *[33] The total number of comparators of Yao's selection network is upper bounded by* $d\lceil\log(k+1)\rceil + c_k \cdot \log d^{\lceil\log((k+1)/3)\rceil}$, *where* $c_k \in O(k^{2+\lambda_k})$ *and* $\lambda_k \in O(\log\log k)$.

For small $k \ll d$, the number of comparators in Yao's selection network is dominated by $O(d \log k)$. Earlier than this, Alekseev proposed a different construction for oblivious top-k network [2], which was later improved by Cong et al. [9]. Cong et al.'s selection network uses $O(d \log^2 k)$ comparators, which outperforms Yao's method for $k \in \Omega(\sqrt{d})$. We choose to stick with Yao's method, as we consider it impractical to apply DP top-k selection with $k \in \Omega(\sqrt{d})$, due to limited privacy budget. [1]

Directly applying a traditional selection network raises a new complication, as the swappers operate on numerical entries, whereas we want the output to be the corresponding indices. To resolve this issue, we pair each numerical entry and the corresponding index into blocks and let the swappers take blocks as input. When a swapper swaps two entries inside the circuit, it also swaps the underlying indices. The inclusion of indices also ensures that ties between equal entries can properly be broken according to Algorithm 2. When the selection network terminates, the top-k indices can be extracted from the output blocks to be fed into the next stage.

[1] In principal, one can adopt a hybrid approach which leverages the strength of both selection networks to achieve optimal efficiency. We choose to neglect this approach due to its implementation complexity, and the efficiency gain is reported to be small especially for small values of k [9].

4.2.3 Shuffling

The previous selection phase computes the top-k indices of $\tilde{x}_1 + \tilde{x}_2$ as in Algorithm 2. These indices cannot be revealed publicly yet, as the privacy proof to Algorithm 2 requires them to be released as an unordered set. It is therefore necessary to shuffle these indices so that their ordering at the end of the selection network becomes unknown. This shuffling procedure is implemented by the Waksman network [30]. The Waksman network takes an array and then applies a permutation to the input array according to some additional control bits. By configuring these control bits, the Waksman network can apply any permutation on the input array. In our setting, this input array is composed of the top-k indices at the end of the selection phase. We let one party choose a random permutation and set control bits to the shuffling network which realizes that permutation. After the top-k indices goes through the shuffling network, the other party receives the permuted output, after which they can sort the result and send them to the first party. Since the first party is unaware of the permuted outcome and the second party is unaware of the permutation applied, neither of them learns the initial ordering of the top-k indices inside the circuit.

The protocol is thus complete when both parties receive the output indices. A complete pseudocode is presented in the next section.

4.2.4 Complete Protocol

We present the protocol in Algorithm 4, with privacy guarantee below.

Algorithm 4 Baseline two-party DP selection

Input: P_1, P_2 each holds private vectors $x_1, x_2 \in \mathbb{N}^d$, $1 \le k \le d, \epsilon > 0$

1: P_1 computes $\tilde{x}_1 \leftarrow x_1 + \eta_1$, where $\eta_1 \sim \text{Geometric}(1 - e^{-\epsilon})$
2: P_2 computes $\tilde{x}_2 \leftarrow x_2 + \eta_2$, where $\eta_2 \sim \text{Geometric}(1 - e^{-\epsilon})$
3: P_1 uniformly samples a permutation π on $\{0, \dots, d-1\}$
4: P_1 configures control bits s to realize π in a Waksman network
5: P_1, P_2 evaluate the subroutine Top-k-Set$(\tilde{x}_1, \tilde{x}_2, s)$ in Algorithm 5 inside a garbled circuit, where P_2 receives the output index set o
6: P_2 sorts o and sends it to P_1

Theorem 4. *Algorithm 4 simulates Algorithm 2 in the presence of computationally bounded semi-honest adversaries.*

Algorithm 5 Top-k-set subroutine

function TOP-k-SET($(\tilde{x}_1, \tilde{x}_2, s)$)
    $A \leftarrow$ array of size d
    for $i = 0, \ldots, d-1$ **do**
        $A[i] = (i, \tilde{x}_1[i] + \tilde{x}_2[i])$
    run Yao's top-k selection network in Algorithm 3 to move the top-k entries of A to its front
    $B \leftarrow$ array of size d
    for $i = 0, \ldots, k-1$ **do**
        $B[i] = A[i]$.first
    run the Waksman network to permute array B using control bits s
    return B

Proof. We prove the result by showing that both parties' view in Algorithm 4 can be simulated by having access to the output of Algorithm 2. Let the output o of Algorithm 2 be given. P_1's view in Algorithm 4 can be simulated by drawing a uniformly random permutation π in line 3, and a sorted result of o in line 6. P_2's view can be simulated by a uniformly permuted result of o in line 5. The security of garbled circuit ensures that every other message that P_1, P_2 see in the protocol can be simulated by drawing uniformly random bits. □

The privacy guarantee of Algorithm 4 now follows from Corollary 1 and Theorem 4.

Corollary 2. *Algorithm 4 provides $k\epsilon$-IND-CDP.*

Chapter 5

Threshold Trimming Algorithm

In this chapter, we present our first heuristic protocol for two-party DP selection that achieves sublinear runtime in the MPC phase. We start by developing a heuristic algorithm in the non-private case and then explain how to adapt it to achieve DP. Finally, we discuss the protocol's implementation in a two-party MPC setting.

5.1 Heuristic in the Non-private Setting

The high-level idea behind our heuristic is to let each party individually select an index set with good local utility, and then constrain the selection domain to the union of those two index sets. To make it more precise, let P_1, P_2 be the two participating parties with private vectors x_1, x_2 respectively. We let P_1 locally computes S_1, which is an index set of large entries in x_1. Define S_2 for P_2 similarly. After that, we run an MPC selection protocol on $(x_1 + x_2)_{S_1 \cup S_2}$. Note that this procedure readily admits a sublinear MPC protocol when $|S_1|, |S_2|$ is sufficiently small compared to the entire domain size d.

A design choice we have to make now is whether each party's locally trimmed selection domain should be kept private. Leaking the selection domain would simplify the MPC process, but would introduce extra privacy budget to the protocol as a tradeoff. We explore both approaches in this thesis. In this chapter, we design a protocol based on the heuristic above, while hiding the trimmed selection domain. This domain approximates the union of top-m index sets from both parties, for some $m \geq k$. The alternative approach, where the selection domain is public, will be examined in the next chapter.

Even in the non-private case, hiding the trimmed selection domain already raises a new challenge in the MPC phase: since a party does not know the entire input domain of the two-party selection procedure, it also cannot know which entries to give as input to the MPC protocol. This problem suggests that each party would need to supply every vector entry, which renders a sublinear MPC protocol impossible. We remedy this issue by asking each party to provide the average of its non-top-m entries. This average value then serves as an estimate for all of its vector entries outside its trimmed domain. Based on this approximation, we arrived at the following pseudocode in Algorithm 6. Note that $x_{(i)}$ stands for the i-th largest entry in x.

Algorithm 6 ∞-DP heuristic

Input: P_1, P_2 hold private vectors $x_1, x_2 \in \mathbb{N}^d$ respectively, $1 \leq k \leq m \leq d, \epsilon > 0, \epsilon' > 0$

$S_1 \leftarrow$ top-m index set in x_1
$S_2 \leftarrow$ top-m index set in x_2
$a_1 \leftarrow \lfloor \frac{1}{d-m} \sum_{i=m+1}^{d} (x_1)_{(i)} \rceil$
$a_2 \leftarrow \lfloor \frac{1}{d-m} \sum_{i=m+1}^{d} (x_2)_{(i)} \rceil$
$s \leftarrow$ all-zero array with size d
for $i \in S_1 \cup S_2$ **do**
    if $i \in S_1 \cap S_2$ **then**
        $s[i] \leftarrow x_1[i] + x_2[i]$
    else if $i \in S_1$ **then**
        $s[i] \leftarrow x_1[i] + a_2$
    else
        $s[i] \leftarrow a_1 + x_2[i]$
    return top-k index set of $s_{S_1 \cup S_2}$ (break ties by preferring small indices)

5.2 Incorporating DP

We now explain how to incorporate DP into Algorithm 6. To simplify things, we focus on party P_1, as the case for P_2 is symmetric. At a high level, the DP version of Algorithm 7 adds geometric noise to certain variables and then executes the non-private algorithm on the noisy values. However, there is an important difference in how these two algorithms define S_1. Whereas the non-private algorithm takes S_1 to be the top-m index set of x_1, the DP algorithm computes a certain threshold T_1 which approximates the $(m+1)$-th largest entry in x_1, and then take S_1 to be the collection of indices whose corresponding noisy

entries exceed T_1. Aside from this difference, we also note that in the DP algorithm, P_1 uses the same noisy vector $\tilde{x}_1$ for both the local trimming stage and the two-party selection stage.

The DP version of Algorithm 6 is detailed in Algorithm 7, which we refer to as the *thresholding trimming algorithm*. In line 1-2, P_1 computes $\tilde{x}_1$ and a_1 by adding i.i.d. geometric noise to its private vector x_1 and the average of x_1's non-top-m entries, truncated to the nearest integer. The party then computes a threshold T_1 in line 3, which is the sum of $\max((\tilde{x}_1)_{(m+1)}, a_1)$ and another geometric noise. In line 4, P_1 computes $S_1 = \{0 \le i < d \mid \tilde{x}_1[i] > T_1\}$, which concludes the local trimming stage. We denote the counterpart of $\tilde{x}_1, a_1, S_1$ from P_2 by $\tilde{x}_2, a_2, S_2$. In the two-party selection stage, both parties securely evaluate a circuit that constructs an array s indexed on $S_1 \cup S_2$, where

$$s[i] = \begin{cases} \tilde{x}_1 + \tilde{x}_2 & (i \in S_1 \cap S_2) \\ \tilde{x}_1 + a_2 & (i \in S_1 \setminus S_2) \\ a_1 + \tilde{x}_2 & (i \in S_2 \setminus S_1) \end{cases} .$$

The circuit then computes the top-k indices in s, or simply returns s if $|S_1 \cup S_2| \le k$. The protocol completes when both parties receive the output indices.

We make two remarks about Algorithm 7 before moving onto its privacy proof. First, for every $i \in S_1$, we have: $\tilde{x}_1[i] > T_1 \ge (\tilde{x}_1)_{(m+1)}$. This ensures that $|S_1| \le m$, and therefore the number of input values to the MPC protocol is in $O(m)$. Second, $|S_1 \cup S_2|$ could be less than k, which means that our protocol may not always return k indices. We do not see a way to circumvent this issue with existing techniques.

5.2.1 Proof of Privacy

Our privacy proof to the threshold trimming algorithm builds upon the idea of *randomness alignment*, which has been applied in many DP algorithms as well as some DP verification tools [12, 1, 31]. Recall that in order to show a mechanism $\mathcal{M}$ is ϵ-DP, we must prove $\mathbb{P}(\mathcal{M}(D) \subseteq E) \le e^{\epsilon}\mathbb{P}(\mathcal{M}(D') \subseteq E)$, where D, D' are two neighboring datasets and E is an arbitrary set. We denote the randomness used in $\mathcal{M}$ by a vector $H \in \mathbb{R}^{\infty}$. With a slight abuse of notation, we use $\mathcal{M}(D, H)$ to refer to the output of algorithm $\mathcal{M}$ on input D with randomness H. Let S be the set of random vectors H that lets $\mathcal{M}$ output an element in E with input D. Define S' for D' similarly. The inequality in the definition of DP may now be written as $\mathbb{P}_{\mathcal{M}}(H \in S) \le e^{\epsilon}\mathbb{P}_{\mathcal{M}}(H \in S')$. The randomness alignment technique proves this inequality by introducing an intermediate quantity: $\mathbb{P}_{\mathcal{M}}(H \in S) \le$

Algorithm 7 Threshold trimming algorithm (ideal functionality)

Input: P_1, P_2 hold private vectors $x_1, x_2 \in \mathbb{N}^d$ respectively, $1 \le k \le m \le d, \epsilon > 0, \epsilon' > 0$

$\tilde{x}_1 \leftarrow x_1 + \eta_1$ where $\eta_1 \sim \text{Geometric}(1 - e^{-\epsilon})^d$
$a_1 \leftarrow \lfloor \frac{1}{d-m} \sum_{i=m+1}^{d} (x_1)_{(i)} \rceil + h_1$, where $h_1 \sim \text{Geometric}(1 - e^{-\epsilon})$
$T_1 \leftarrow \max((\tilde{x}_1)_{(m+1)}, a_1) + g_1$, where $g_1 \sim \text{Geometric}(1 - e^{-\epsilon'})$
$S_1 \leftarrow \{0 \le i < d \mid \tilde{x}_1[i] > T_1\}$
compute $\tilde{x}_2$, a_2, S_2 similarly for P_2
$s \leftarrow$ all-zero array with size d
for $i \in S_1 \cup S_2$ **do**
    if $i \in S_1 \cap S_2$ **then**
        $s[i] \leftarrow \tilde{x}_1[i] + \tilde{x}_2[i]$
    else if $i \in S_1$ **then**
        $s[i] \leftarrow \tilde{x}_1[i] + a_2$
    else
        $s[i] \leftarrow \tilde{x}_2[i] + a_1$
if $|S_1 \cup S_2| \ge k$ **then**
    return top-k index set in $s_{S_1 \cup S_2}$ (break ties by preferring small indices)
else
    return $S_1 \cup S_2$

$e^\epsilon \mathbb{P}_{\mathcal{M}}(H \in \Phi(S)) \leq \mathbb{P}_{\mathcal{M}}(H \in S')$, where $\Phi : \mathbb{R}^\infty \to \mathbb{R}^\infty$ is an injective function such that $\mathcal{M}(D, H) = \mathcal{M}(D', \Phi(H))$ for all H [1]. In other words, applying Φ on randomness H preserves the output of $\mathcal{M}$ when the underlying dataset changes from D to D'. This function Φ is also sometimes referred to as the alignment function. Once an alignment function Φ has been found, the second part of the inequality follows immediately from that

$$\Phi(S) = \{\Phi(H) \mid \mathcal{M}(D, H) \in E\} = \{\Phi(H) \mid \mathcal{M}(D', \Phi(H)) \in E\} \subseteq S',$$

whereas the first part of the inequality can be established by computing the ratio of probability densities between H and $\Phi(H)$. This completes the DP proof. In fact, our proof to Theorem 2 is also an application of this technique.

We now use the randomness alignment technique to establish the privacy guarantee for Algorithm 7. The bulk of the proof is to show that the alignment function Φ satisfies $\mathcal{M}(D, H) = \mathcal{M}(D', \Phi(H))$ for all H.

Theorem 5. *Algorithm 7 satisfies $(k\epsilon + \epsilon')$-DP.*

Proof. Fix $k, m, \epsilon, \epsilon'$. Let Algorithm 7 be denoted by $\mathcal{M}$. Since $\mathcal{M}$ is symmetric with respect to P_1 and P_2, it suffices to show that $\mathcal{M}$ is $(k\epsilon + \epsilon')$-DP for P_1. This also lets us fix $\tilde{x}_2, a_2$ and S_2. We may now think of $\mathcal{M}$ as a random function of x_1 and write $\mathcal{M}(x_1)$. Borrowing from the earlier discussion on randomness alignment, we also use $\mathcal{M}(x_1, (\eta_1, h_1, g_1))$ to denote the output of $\mathcal{M}$ on x_1 under fixed random variables η_1, h_1, g_1, which are used in line 1-3.

Let $f : \mathbb{N}^d \to \mathbb{N}$ be the truncated average of the non-top-m entries in the input vector, as used in line 2.

Let $x_1, x_1' \in \mathbb{N}^d$ where $x_1 \sim x_1'$, and o be a possible output. We will also assume that x_1' is obtained from x_1 by adding a user vector. Equivalently, $x_1' - x \in \{0, 1\}^d$. The opposite case for removing a user vector can be reduced to the previous case. This reduction is described at the end of the proof.

We will prove the stronger result that the algorithm remains DP even if it additionally leaks $o \cap S_1$. Suppose that $\mathcal{M}(x_1, (\eta, h, g))$ outputs o and $o \cap S_1$. Write $E = o \cap S_1$. We

[1] In continuous domain, one would also need to verify that the Jacobian of Φ is continuous almost everywhere [12]. This does not concern us as our algorithm only uses discrete probability distributions.

claim that $\mathcal{M}(x_1', (\eta', h', g'))$ also outputs o and E, where

$$\eta' \coloneqq \eta + \mathbb{1}_E \tag{5.1}$$

$$h' \coloneqq h + \begin{cases} 0 & (E = o) \\ f(x_1) - f(x_1') + 1 & (E \subsetneq o) \end{cases} \tag{5.2}$$

$$g' \coloneqq g + \max((x_1 + \eta)_{(m+1)}, a_1) - \max((x_1' + \eta)_{(m+1)}, a_1') + 1. \tag{5.3}$$

In the following discussion, we use (o, E) and (o', E') to indicate the output pairs of $\mathcal{M}(x_1, (\eta, h, g))$ and $\mathcal{M}(x_1', (\eta', h', g'))$ respectively. The goal here is to show that $(o, E) = (o', E')$.

Before proceeding with the proof, we must first verify that η', h', g' are all non-negative. It is straightforward to see that this is true for η' and h'. As for g', observe that $(x_1 + \eta)_{(m+1)} - (x_1' + \eta)_{(m+1)} \geq -1$ and

$$a_1 - a_1' = f(x_1) + h - (f(x_1') + h') = \begin{cases} f(x_1) - f(x_1') & (E = o) \\ -1 & (E \subsetneq o) \end{cases} \geq -1.$$

Therefore, $\max((x_1 + \eta)_{(m+1)}, a_1) - \max((x_1' + \eta)_{(m+1)}, a_1') \geq -1$, which follows $g' \geq 0$.

We now lay out some notation rules for the rest of the discussion. We use $\tilde{x}_1, a_1, T_1, S_1, s$ to denote the corresponding variables in Algorithm 7, during the execution of $\mathcal{M}(x_1, (\eta, h, g))$. We also use $\tilde{x}_1', a_1', T_1', S_1', s'$ to denote the counterpart of these variables during the execution of $\mathcal{M}(x_1', (\eta', h', g'))$. A list of definitions is provided in Table 5.1.

Variable	Definition
$x_1, x_1', \eta, h, g, \tilde{x}_2, a_2, S_2$	pre-given variables
$\tilde{x}_1$	$x_1 + \eta$
a_1	$f(x_1) + h$
T_1	$\max((\tilde{x}_1)_{(m+1)}, a_1) + g$
S_1	$\{0 \le i < d \mid \tilde{x}_1[i] > T_1\}$
s (indexed on $S_1 \cup S_2$)	$s[i] = \begin{cases} \tilde{x}_1[i] + \tilde{x}_2[i] & (i \in S_1 \cap S_2) \\ \tilde{x}_1[i] + a_2 & (i \in S_1 \setminus S_2) \\ a_1 + \tilde{x}_2[i] & (i \in S_2 \setminus S_1) \end{cases}$
o	top-k index set of s
E	$o \cap S_1$
η', h', g'	defined from η, h, g as in Equation (5.1) - (5.3)
$\tilde{x}_1'$	$x_1' + \eta'$
a_1'	$f(x_1') + h'$
T_1'	$\max((\tilde{x}_1')_{(m+1)}, a_1') + g'$
S_1'	$\{0 \le i < d \mid \tilde{x}_1'[i] > T_1'\}$
s' (indexed on $S_1' \cup S_2$)	$s[i] = \begin{cases} \tilde{x}_1'[i] + \tilde{x}_2[i] & (i \in S_1' \cap S_2) \\ \tilde{x}_1'[i] + a_2 & (i \in S_1' \setminus S_2) \\ a_1' + \tilde{x}_2[i] & (i \in S_2 \setminus S_1') \end{cases}$
o'	top-k index set of s'
E'	$o' \cap S_1'$

Table 5.1: Variable definitions in the proof of Theorem 5

The first step of the proof is to establish that $T_1' = T_1 + 1$. This requires the following lemma.

Lemma 3. $(\tilde{x}_1')_{(m+1)} = (x_1' + \eta)_{(m+1)}$.

Proof. We rewrite the lemma above as $(x_1' + \eta + \mathbb{1}_E)_{(m+1)} = (x_1' + \eta)_{(m+1)}$. Let $K = \{j \mid \tilde{x}_1[j] \le (\tilde{x}_1)_{(m+1)}\}$. Since $E \subseteq S_1 \subseteq \{i \mid \tilde{x}_1[i] > (\tilde{x}_1)_{(m+1)}\}$, E is disjoint from K. For all $i \in E$ and $j \in K$, we have:

$$(x_1' + \eta)[i] \ge (x_1 + \eta)[i] = \tilde{x}_1[i] \underbrace{\ge}_{i \in E} (\tilde{x}_1)_{(m+1)} + 1 \underbrace{\ge}_{j \in K} \tilde{x}_1[j] + 1 \ge (x_1' + \eta)[j].$$

This tells us that we may rank entries in $x'_1 + \eta$ by descending order in a way such that every index in E comes before every index in K. Since $|K| \geq d - m$, every index in E has rank at most m in $x'_1 + \eta$. As a result, adding $\mathbb{1}_E$ onto $x'_1 + \eta$ does not change its $(m+1)$-th largest entry, which proves the lemma. □

Back to the original problem, we have:

$$\begin{aligned}
T'_1 =& \max((\tilde{x}'_1)_{(m+1)}, a'_1) + g' \\
=& \max((x'_1 + \eta)_{(m+1)}, a'_1) + g' \qquad \text{(Lemma 3)} \\
=& \max((x'_1 + \eta)_{(m+1)}, a'_1) + g + \max((x_1 + \eta)_{(m+1)}, a_1) \\
& - \max((x'_1 + \eta)_{(m+1)}, a'_1) + 1 \\
=& g + \max((x_1 + \eta)_{(m+1)}, a_1) + 1 \\
=& T_1 + 1,
\end{aligned}$$

as required.

The next step of the proof is to show $E \subseteq S'_1 \subseteq S_1$. The first inclusion is true because for all $i \in E$,

$$\begin{aligned}
i \in E &\implies \tilde{x}_1[i] > T_1 \qquad (i \in E \subseteq S_1) \\
&\implies \tilde{x}_1[i] + 1 > T'_1 \qquad (T'_1 = T_1 + 1) \\
&\implies \tilde{x}'_1[i] > T'_1 \\
&\implies i \in S'_1.
\end{aligned}$$

Furthermore, for all $j \in \{0, \ldots, d-1\} \setminus S_1$, we have:

$$\begin{aligned}
j \notin S_1 &\implies \tilde{x}_1[j] \leq T_1 \\
&\implies \tilde{x}_1[j] + 1 \leq T'_1 \qquad (T'_1 = T_1 + 1) \\
&\implies \tilde{x}'_1[j] \leq T'_1 \qquad (j \notin S_1 \supseteq E) \\
&\implies j \notin S'_1,
\end{aligned}$$

which proves $S'_1 \subseteq S_1$.

The fact that $E \subseteq S'_1 \subseteq S_1$ directly implies $(o, E) = (o', E')$ when $|S_1 \cup S_2| \leq k$. If $|S_1 \cup S_2| \leq k$, then $o = S_1 \cup S_2$, and $E = o \cap S_1 = S_1$. Substituting E with S_1 in $E \subseteq S'_1 \subseteq S_1$ yields $S_1 \subseteq S'_1 \subseteq S_1$, which translates to $S'_1 = S_1$. As a result, $|S'_1 \cup S_2| \leq k$,

and it follows that $o' = S_1' \cup S_2 = S_1 \cup S_2 = o$ and $E' = S_1' = S_1 = E$, which is exactly what we want. Hence, we may thereafter assume $|S_1 \cup S_2| > k$.

We now show that o is the top-k index set of s' (indexed on $S_1' \cup S_2$), where ties are broken by preferring smaller indices. Since $o \cap S_1 = E \subseteq S_1'$, we can be certain that o is indeed a subset of $S_1' \cup S_2$. We claim that for all $i \in o, j \in (S_1' \cup S_2) \setminus o$, $s'[i] \geq s[i] + 1$ and $s'[j] \leq s[j] + 1$.

For notational convenience, let $y[i] = \begin{cases} \tilde{x}_2[i] & (i \in S_2) \\ a_2 & (i \notin S_2) \end{cases}$ for all $0 \leq i < d$. This definition lets us write $s[i] = y[i] + \begin{cases} \tilde{x}_1[i] & (i \in S_1) \\ a_1 & (i \notin S_1) \end{cases}$ and $s'[i] = y[i] + \begin{cases} \tilde{x}_1'[i] & (i \in S_1') \\ a_1' & (i \notin S_1') \end{cases}$.

Let $i \in o$. If $i \in S_1$, then $i \in o \cap S_1 = E$ and $i \in E \subseteq S_1'$. We have:

$$s'[i] \underbrace{=}_{i \in S_1'} \tilde{x}_1'[i] + y[i] \underbrace{=}_{i \in E} (x_1'[i] + \eta[i] + 1) + y[i] \geq (x_1[i] + \eta[i] + y[i]) + 1 \underbrace{=}_{i \in S_1} s[i] + 1,$$

which is what we want. If $i \notin S_1$, then we also have:

$$\begin{aligned} s'[i] &= a_1' + y[i] \quad (i \notin S_1 \supseteq S_1') \\ &= (f(x_1') + h') + y[i] \\ &= f(x_1') + (h + f(x_1) - f(x_1') + 1) + y[i] \quad (i \notin E \implies E \subsetneq o) \\ &= (f(x_1) + h + y[i]) + 1 \\ &= s[i] + 1 \quad (i \notin S_1). \end{aligned}$$

In conclusion, $s'[i] \geq s[i] + 1$ whenever $i \in o$. This proves the first half of our claim.

Let $j \in (S_1' \cup S_2) \setminus o$. If $j \in S_1'$, then j must also be in S_1 and

$$s'[j] \underbrace{=}_{j \in S_1'} \tilde{x}_1'[j] + y[j] \underbrace{=}_{j \notin o \supseteq E} x_1'[j] + \eta[j] + y[j] \leq x_1[j] + \eta[j] + 1 + y[j] \underbrace{=}_{j \in S_1} s[j] + 1.$$

Suppose that $j \notin S_1'$. We have two cases: (a) $j \notin S_1$ and (b) $j \in S_1$. In the first case,

$$s'[j] \underbrace{=}_{j \notin S_1'} f(x_1') + h' + y[j] \leq f(x_1') + (h + f(x_1) - f(x_1') + 1) + y[j] = (f(x_1) + h + y[j]) + 1 \underbrace{=}_{j \notin S_1} s[j] + 1.$$

In the second case,

$$\begin{aligned} s'[j] &= a'_1 + y[j] \quad (j \notin S'_1) \\ &\leq T'_1 + y[j] \\ &= (T_1 + 1) + y[j] \\ &< \tilde{x}_1[j] + 1 + y[j] \quad (j \in S_1 \implies \tilde{x}_1[j] > T_1) \\ &= s[j] + 1. \end{aligned}$$

We have now shown that $s'[j] \leq s[j] + 1$ for all $j \in S'_1 \cup S_2 \setminus o$.

Now, for all $i \in o$ and $j \in (S' \cup S_2) \setminus o$, $s'[i]$ must dominate $s'[j]$ because

$$s'[j] - s'[j] \geq (s[i] + 1) - (s[j] + 1) = s[i] - s[j] \geq 0.$$

Furthermore, if $s'[i] = s'[j]$, then we must also have $s[i] = s[j]$, which follows $i < j$, for otherwise j would have been in o instead of i. This implies that o is the top-k index set in s'. In other words, $o' = o$.

It is now straightforward to show $E' = E$. Recall that $o \cap S_1 = E \subseteq S'_1 \subseteq S_1$. Take the intersection between o and each item in this chain of inclusion, and we have:

$$o \cap (o \cap S_1) \subseteq o \cap S'_1 \subseteq o \cap S_1.$$

Equivalently, $E \subseteq E' \subseteq E$, which follows $E' = E$. Combine everything together, and we conclude that that $\mathcal{M}(x_1, (\eta, h, g)) = (o, E) = (o', E') = \mathcal{M}(x'_1, (\eta', h', g'))$, as required.

We finish off the proof by invoking the randomness alignment technique. Let Φ be the mapping $[(\eta, h, g) \mapsto (\eta', h', g')]$, where η', h', g' are defined from Equation (5.1) - (5.3) as before. Note that Φ is an injection. By the property of geometric distribution, we have:

$$\frac{\mathbb{P}((\eta, h, g))}{\mathbb{P}(\Phi(\eta, h, g))} \leq \exp(\epsilon \cdot (\|\eta - \eta'\|_1 + |h - h'|) + \epsilon' \cdot |g - g'|),$$

where $\|\eta - \eta'\|_1 \leq k, |h - h'| \leq 1, |g - g'| \leq 1$. Furthermore, $\|\eta - \eta'\|_1 < k$ when $|h - h'| = 1$, because

$$|h - h'| = 1 \implies E \subsetneq o \implies |E| < k \implies \|\eta - \eta'\|_1 < k.$$

This implies that $\mathbb{P}((\eta, h, g))/\mathbb{P}(\Phi(\eta, h, g)) \leq \exp(k\epsilon + \epsilon')$.

Let (o, E) be a possible output pair of $\mathcal{M}$, and let $\mathcal{S} = \{(\eta, h, g) \mid \mathcal{M}(x_1, (\eta, h, g)) = (o, E)\}$. Define $\mathcal{S}'$ for x'_1 similarly. Since $\mathcal{M}(x_1, (\eta, h, g)) = \mathcal{M}(x'_1, \Phi(\eta, h, g))$ for all η, g, h,

$\Phi(\mathcal{S}) \subseteq \mathcal{S}'$. We have:

$$\begin{aligned}
\mathbb{P}(\mathcal{M}(x_1) = (o, E)) &= \sum_{(\eta,h,g)\in\mathcal{S}} \mathbb{P}_{\mathcal{M}}((\eta, h, g)) \\
&\leq \exp(k\epsilon + \epsilon') \sum_{(\eta,h,g)\in\mathcal{S}} \mathbb{P}_{\mathcal{M}}(\Phi(\eta, h, g)) \\
&\leq \exp(k\epsilon + \epsilon') \sum_{(\eta,h,g)\in\mathcal{S}'} \mathbb{P}_{\mathcal{M}}((\eta, h, g)) \quad (\Phi(\mathcal{S}) \subseteq \mathcal{S}') \\
&= \exp(k\epsilon + \epsilon') \cdot \mathbb{P}(\mathcal{M}(x_1') = (o, E)),
\end{aligned}$$

which proves our result.

We sketch the proof for the case where x_1' is obtained by subtracting a user vector from x_1. Observe that $\mathcal{M}(x_1)$ is identically distributed as $\mathcal{M}(x_1 - \mathbb{1})$. As such, for any possible output set E, we have: $\mathbb{P}(\mathcal{M}(x_1) \in E) = \mathbb{P}(\mathcal{M}(x_1 - \mathbb{1}) \in E) \leq e^{k\epsilon+\epsilon'}\mathbb{P}(\mathcal{M}(x_1') \in E)$, where the second inequality comes from our DP proof to the previous case. This argument fails when x_1 has a zero entry, but in this scenario we can add $\mathbb{1}$ to both x_1 and x_1' and proceed the argument as before. □

We make a final remark that in order for Theorem 5 to hold, it is crucial that the same noisy vector $\tilde{x}_1$ (and $\tilde{x}_2$ for P_2) is used in both the local trimming stage at line 3-4 and the two-party selection stage at line 7-17. Indeed, if the algorithm refreshes the noise vector between the two stages, then the alignment function Φ in the privacy proof would need to perturb both noise vectors in order to preserve the algorithm's output. This would cause Φ to further scale down the probability mass of a given randomness vector, resulting in more privacy budget than what we currently need.

5.3 MPC Implementation

Our MPC implementation of Algorithm 7 translates its pseudocode into a garbled circuit. Since the algorithm implicitly computes $S_1 \cap S_2$, the protocol resembles closely to the *sort-compare-shuffle* paradigm in private set intersection. The parties start by locally computing $\tilde{x}_1, a_1, S_1$ and $\tilde{x}_2, a_2, S_2$, respectively for P_1 and P_2. Next, the parties send a_1, a_2, as well as $(\tilde{x}_1)_{S_1}$ and $(\tilde{x}_2)_{S_2}$, each entry with its value, index, and an extra bit indicating the input party, into the garbled circuit. Since $|S_1|, |S_2|$ are not predetermined, the parties pad the input arrays $(\tilde{x}_1)_{S_1}, (\tilde{x}_2)_{S_2}$ with dummy items to achieve a constant

array size of m. The garbled circuit then runs an odd-even merging network [4] to sort the union of the two arrays by indices of these entries. After that, the circuit compares neighboring elements in the union array to construct s which approximates $(\tilde{x}_1 + \tilde{x}_2)_{S_1 \cup S_2}$ (Section 5.3.1), as in line 7-17. The array s can then be fed into a selection network to produce the desired top-k index set. This last step is the same as in our baseline protocol in Algorithm 4.

5.3.1 Constructing the Sum Array

The oblivious merging network produces an array of blocks, each of which represents an entry in $\tilde{x}_1$ or $\tilde{x}_2$. Each block contains the entry's value and index, and an additional bit indicating which party it comes from. The merging procedure ensures that the blocks are sorted by corresponding indices. After this procedure, we iterate over each block in the for-loop in line 3. For each block B, we first check if its index i is in $S_1 \cap S_2$, by examining if i matches B's neighboring entry's index. If i is in the intersection, then we create the corresponding entry in s, with its value $\tilde{x}_1[i] + \tilde{x}_2[i]$. This is equal to the sum of B's and its neighbor's values. If i is not in the intersection, then we check if B comes from P_1. If the answer is yes, then the value in B corresponds to $\tilde{x}_1[i]$, in which case we add a_2 to it and create the corresponding entry in s. The opposite case when B comes from P_2 is handled similarly, by adding a_1 to B's value and repeating the same. This procedure is detailed in Algorithm 8.

Although the pseudocode in Algorithm 8 does not describe a static circuit, it is not difficult to incorporate it in a garbled circuit protocol. This is due to the fact that in each iteration of the main loop in line 3, the program only accesses a constant number of data-independent memory locations. As a result, the program can be efficiently translated into a Boolean circuit using standard techniques in MPC. We do not elaborate on further details of the circuit design, as it is less relevant for our study.

After the algorithm returns s, we send s into Yao's selection network to produce its top-k index set o. To hide the ordering information of indices in o, we apply a Waksman permutation network on o, with its permutation supplied by P_1. The permuted result can then be revealed to P_2, after which P_2 can sort it and send it back to P_1, thus completing the protocol.

We present a complete pseudocode of the protocol in Algorithm 9, with its privacy guarantee below.

Theorem 6. *Algorithm 9 is $(k\epsilon + \epsilon')$-IND-CDP.*

Algorithm 8 Constructing the Sum Array

Input: a_1, a_2, and array *entries* of size $2m$, each element containing three fields $value, index, isFromP1$. We assume elements in *entries* are sorted by the *index* field. Dummy items have $index = -1$.

$s \leftarrow$ empty array of size $2m$, each element contains two zero-initialized fields $value, index$
$lastMatched \leftarrow$ **false**
for $i \leftarrow 0, \ldots, 2m-1$ **do**
    $thisMatched = (i+1 < 2m$ and $entries[i].index == entries[i+1].index)$
    if $lastMatched$ is **false then** ▷ *Avoids creating the same entry twice in s*
        $s[i].index \leftarrow i$
        if $thisMatched$ is **true then**
            $s[i].value \leftarrow entries[i].value + entries[i+1].value$
        else if $entries[i].isFromP1$ is **true then**
            $s[i].value \leftarrow entries[i].value + a_2$
        else
            $s[i].value \leftarrow entries[i].value + a_1$
    $lastMatched \leftarrow thisMatched$
return s

Algorithm 9 Threshold trimming algorithm (two-party protocol)

Input: P_1, P_2 hold respective private vectors $x_1, x_2 \in \mathbb{N}^d$, $1 \le k \le m \le d, \epsilon > 0, \epsilon' > 0$

1: **function** CIRCUIT(A_1, a_1, A_2, a_2, C)
2: use Batcher's odd-even merging network to sort $A = A_1 \cup A_2$
3: run Algorithm 8 on a_1, a_2, A to construct array s
4: run Yao's top-k selection network to compute the top-k index set o of s
5: apply the Waksman network on o using control bits C
6: **return** o
7: P_1 locally computes $\tilde{x}_1, a_1, S_1$ as in Algorithm 7
8: P_2 computes $\tilde{x}_2, a_2, S_2$ similarly
9: P_1 initializes array $A_1 = \{(\tilde{x}_1[i], i, 1)\}_{i \in S_1}$, padded with dummy items to size m, pre-sorted by index i
10: P_2 initializes array $A_2 = \{(\tilde{x}_2[i], i, 0)\}_{i \in S_2}$, padded with dummy items to size m, pre-sorted by index i
11: P_1 uniformly samples a permutation on $\{0, \ldots, k-1\}$ and configures the corresponding control bits C in a Waksman network
12: both parties evaluate Circuit(A, a_1, A, a_2, C), where P_2 first receives the output o
13: P_2 sorts o and sends it back to P_1

Proof. Given access to the output o of Algorithm 7, we can simulate the views of P_1 and P_2 in Algorithm 9 as follows. P_1's view can be simulated by drawing a uniformly random permutation on $\{0, \ldots, k-1\}$ in line 11 and a sorted result of o in line 13. P_2's view can simulated by applying a uniformly sampled permutation on o in line 13. The security of garbled circuit ensures that every other message that P_1, P_2 see in the protocol is computationally indistinguishable from uniformly random bits. Since P_1, P_2's views in the protocol can be simulated from the output o of Algorithm 7 in a computationally indistinguishable manner, and Algorithm 7 satisfies $(k\epsilon + \epsilon')$-DP (Theorem 5), the entire protocol is $(k\epsilon + \epsilon')$-IND-CDP as required. □

Chapter 6

Joint EM Trimming Algorithm

In this chapter, we explore a different method to apply the local trimming heuristic to the two-party DP selection problem. The main approach is to let each party release an index set with good local utility, and then apply the baseline selection protocol on the union of the two index sets. The problem thus reduces to defining a specific trimming algorithm that lets each party select such an index set. In the previous protocol, we let each party locally compute their approximate top-m index set for some $m \geq k$. We normally want m to be notably larger than k in order to not miss out on strong potential candidates in the two-party selection stage. With this approach however, releasing the approximate top-m index set publicly may require more privacy budget than allowed to produce reasonably useful output. Therefore, we propose to relax the goal of selecting top-m indices at the local trimming stage, in order to alleviate the privacy budget problem.

6.1 Relaxing the Selection Problem

We start by describing the relaxed version of the selection problem as mentioned above. The key idea is that instead of requiring all selected indices to have good local utilities, we only require a portion of them to be good. This idea is captured by the definition below.

Definition 4. *Given a vector $x \in \mathbb{N}^d$, and let $1 \leq m < s \leq d$. The top-m-in-s selection problem asks for an unordered index set S of size s, such that S contains some[1] top-m index set of x.*

[1] x may have multiple top-m index sets due to equal entries. We do not consider tie-breaking in this situation.

In the context of local trimming, m is the number of local top indices that we want the released index set (from each party) to contain, and s is the size of the entire index set. Note that both m and s are hyperparameters in the protocol. The intuition behind top-m-in-s is that finding a set that contains the top-m indices should be easier than finding the exact top-m indices. In the extreme scenario where $s = d$, this problem can be trivially solved by returning the entire index domain $\{0, \ldots, d-1\}$.

6.2 Applying Exponential Mechanism

Our solution to the top-m-in-s problem employs the exponential mechanism. Recall that the exponential mechanism samples o from candidate set O with probability $\mathbb{P}(o) \propto \exp(\epsilon \cdot u(o, D)/(2\Delta))$, where $u(o, D)$ is the utility of o on private dataset D, and Δ is the sensitivity of u. Given count vector x and index set S, we use the following utility function u^* to penalize how far away is S from containing a top-m index set of x:

$$u^*(x, S) = - \min_{x' | S \text{ contains some top-}m \text{ index set of } x'} d(x, x'),$$

where $d(x, x')$ is the minimum number of user vector additions/removals required to change x into x'. This general definition of utility based on dataset distance has been applied to many other DP problems before [26, 16].

The definition of u^* directly implies the following lemma.

Lemma 4. *u^* has sensitivity 1.*

Proof. Let $x, x' \in \mathbb{N}^d$ be two neighboring count vectors, and let $S \in \binom{\{0,\ldots,d-1\}}{s}$. Let $y \in \mathbb{N}^d$ be such that S contains some top-m index set in y, and $u^*(x, S) = -d(x, y)$. We have:

$$u^*(x', S) \geq -d(x', y) \geq -(d(x, y) + d(x, x')) = u^*(x, S) - 1.$$

A similar argument shows that $u^*(x', S) \leq u^*(x, S) + 1$. This concludes our proof. □

Let **JOINT** be the distribution in this instance of exponential mechanism with utility u^*, which we would like to sample from. It remains to be shown how this sampling procedure can be done in a reasonably efficient manner, as the total number of candidates is in $O(d^s)$. Indeed, a sample space of super-polynomial size is the main obstacle in applying the exponential mechanism in many DP problems. We develop an efficient sampling method for our use case, which is explained in the next section.

6.3 Efficient Sampling

To explain the sampling procedure, we start by simplifying the formula for u^*. The result is Lemma 5. In the following discussion, we fix $x \in \mathbb{N}^d, 1 \le m \le s \le d, S \in \binom{\{0,\dots,d-1\}}{s}$ as in Definition 4. We also use $-S$ to denote $\{0,\dots,d-1\} \setminus S$.

Lemma 5. $u^*(x,S) = -\max(0, \max(x_{-S}) - (x_S)_{(m)})$.

Proof. Suppose that S contains some top-m index set of x. Observe that $\max(x_{-S}) \le x_{(m)} = (x_S)_{(m)}$. We have:

$$u^*(x,S) \underbrace{=}_{\text{by definition of } u^*} 0 \underbrace{=}_{\max(x_{-S}) \le x_{(m)}} -\max(0, \max(x_{-S}) - (x_S)_{(m)}),$$

which is what we want.

Suppose that S does not contain any top-m index set in x, then $\max(x_{-S}) > (x_S)_{(m)}$. The lemma now simplifies to $u^*(x,S) = (x_S)_{(m)} - \max(x_{-S})$. We prove this by showing that neither side is greater than the other.

Let x' be the vector obtained from x by adding $\max(x_{-S}) - (x_S)_{(m)}$ to each entry in S. We have:

$$x'_{(m)} \ge (x'_S)_{(m)} = (x_S)_{(m)} + (\max(x_{-S}) - (x_S)_{(m)}) = \max(x_{-S}) = \max(x'_{-S}).$$

The fact that $x'_{(m)} \ge \max(x'_{-S})$ implies S must contain some top-m index set of x'. As a result, $u^*(x,S) \ge -d(x,x') = (x_S)_{(m)} - \max(x_{-S})$. This completes the first half of the proof.

Now, let x' be a vector in $\mathbb{N}^d$ such that S contains some top-m index set of x'. Define $u(v) \coloneqq \max(v_{-S}) - (v_S)_{(m)}$. Observe that u has sensitivity 1. To see this, note that each of $\max(v_{-S})$ and $-(v_S)_{(m)}$ has sensitivity 1, but they cannot change in the same direction. We now have:

$$d(x,x') \underbrace{\ge}_{u \text{ has sensitivity } 1} |u(x) - u(x')| \underbrace{=}_{u(x)>0=u(x')} u(x) = \max(x_{-S}) - (x_S)_{(m)}.$$

Since x' is arbitrary, the definition of u^* implies $u^*(x,S) \le (x_S)_{(m)} - \max(x_{-S})$. Combining both inequalities proves the lemma. □

We now return to the sampling problem. Our sampling procedure builds upon the sampling framework proposed by Gillenwater et al. [16], which is explained below. The sampling framework first partition the candidate set into $\{X_j\}_{j \in I}$ such that for all j, every candidate in X_j has the same utility u_j. We then sample $a \in I$ by the distribution $\mathbb{P}(a) \propto |X_a| \exp(\epsilon u_a/2)$. [2] After we sampled a, we simply return a uniformly random element in X_a.

The problem thus reduces to finding a suitable partition to realize the sampling framework above. Since Lemma 5 implies that the value of $u^*(x, S)$ can be determined from $\max(x_{-S})$ and $(x_S)_{(m)}$, we partition the candidate set $\binom{\{0,\ldots,d-1\}}{s}$ by the rank of these two elements in a sorted array. Let σ be a permutation that ranks x in descending order, with $x[\sigma(1)] \geq x[\sigma(2)] \geq \cdots \geq x[\sigma(d)]$. For all $1 \leq i, j \leq d$, let $C_{i,j}$ be the collection of index set S such that $\max(x_{-S})$ has rank i and $(x_S)_{(m)}$ has rank j (rank 1 stands for the largest entry and so on), where ties in x are broken by σ. To further simplify the partition scheme, we differentiate between two cases: (a) $i > j$ and (b) $i < j$.

Consider case (a). Let $S \in C_{i,j}$ where $i > j$. Observe that S contains $\sigma(1), \ldots, \sigma(m)$, and $u^*(x, S) = 0$. Since every index set in this case has the same utility, we can group all $C_{i,j}$ under this case in the partition scheme. The union of all $C_{i,j}$ where $i > j$ is given by

$$C \coloneqq \{\sigma(\{1, \ldots, m\} \cup P) \mid P \in \binom{\{m+1, \ldots, d\}}{s-m}\}. \tag{6.1}$$

We make a few observations about $C_{i,j}$ in case (b). Let $S \in C_{i,j}$ where $i < j$. Since $(x_S)_{(m)}$ is ranked after $\max(x_{-S})$, S does not contain at least one of $\sigma(1), \ldots, \sigma(m)$. This implies $i \leq m < j$. In addition, $u^*(x, S) = x[\sigma(j)] - x[\sigma(i)]$. Finally, $C_{i,j}$ can be written in the following way:

$$\begin{aligned} C_{i,j} = \{\sigma(\{1, \ldots, i-1\} \cup P \cup \{j\} \cup Q) \mid \\ P \in \binom{\{i+1, \ldots, j-1\}}{m-i}, Q \in \binom{\{j+1, \ldots, d\}}{s-m}\}. \end{aligned} \tag{6.2}$$

After we have determined the partition scheme to be used in the sampling procedure, all that is left is to count the size of each partition and determine how to uniformly sample from each partition. Counting the size of C and any $C_{i,j}$ where $i < j$ can be done by simply counting the number of choices of P, Q in Equation (6.1) and (6.2). Likewise, uniformly sampling from C and $C_{i,j}$ can also be done by uniformly sampling P or P, Q in Equation

[2] When u_a is large, the Gumbel-max-trick provides a numerically stable procedure for this sampling step [18].

(6.1) and (6.2) respectively. This enables us to use the partition scheme to realize the sampling framework on **JOINT**.

A pseudocode of our sampling procedure is provided in Algorithm 10, with runtime and privacy guarantee below.

Algorithm 10 Exponential Mechanism for top-m-in-s

Input: $x \in \mathbb{N}^d$, $1 \le m \le s \le d, \epsilon > 0$

1: sort x in descending order, $x_{\sigma(1)} \ge \cdots \ge x_{\sigma(d)}$
2: $\mathcal{C} \leftarrow \{(0,0)\}$ ▷ *collection of all partitions*
3: $u[0,0] \leftarrow 0$ ▷ *utility of candidates in C*
4: $s[0,0] \leftarrow \binom{d-m}{s-m}$ ▷ *size of C*
5: **for** $i = 1, \ldots, m$ **do**
6: **for** $j = m+1, \ldots, d-(s-m)$ **do**
7: $\mathcal{C} \leftarrow \mathcal{C} \cup \{(i,j)\}$
8: $u[i,j] \leftarrow x[\sigma(j)] - x[\sigma(i)]$ ▷ *utility of candidates in $C_{i,j}$*
9: $s[i,j] \leftarrow \binom{j-i-1}{m-i}\binom{d-j}{s-m}$ ▷ *size of $C_{i,j}$*
10: sample X from $\mathcal{C}$ with $\mathbb{P}(X) \propto s[i,j]\exp(\epsilon \cdot u[i,j]/2)$
11: **if** $X = (0,0)$ **then**
12: sample P uniformly from $\binom{\{m+1,\ldots,d\}}{s-m}$
13: return $\sigma(\{1,\ldots,m\} \cup P)$
14: **else**
15: write $X = (i,j)$
16: sample P, Q uniformly from $\binom{\{i+1,\ldots,j-1\}}{m-i}, \binom{\{j+1,\ldots,d\}}{s-m}$ respectively
17: **return** $\sigma(\{1,\ldots,i-1\} \cup P \cup \{j\} \cup Q)$

Theorem 7. *Algorithm 10 runs in $O(md)$ time.* [3]

Proof. The runtime of Algorithm 10 is dominated by line 5 - 10, whereas every other line takes at most $O(d)$ time. The nested loop at line 5 takes $O(md)$ time, and sampling from $\mathcal{C}$ at line 10 can also be done in $O(md)$ time, as $\mathcal{C}$ has size at most $O(md)$. Therefore, the runtime of the entire algorithm is in $O(md)$. □

Theorem 8. *Algorithm 10 is ϵ-DP.*

[3] It is possible to make this sampling step more efficient by applying Hao and Zhang's improvement [19] on the original JOINT mechanism by Gillenwater et al. [16]. This technique truncates the utility function to a lower bound so that a large number of partitions can be grouped together in the sampling step, as the candidates in these partitions share the same utility value after truncation. We do not consider this approach as our goal is to reduce the complexity in the multi-party computation phase.

Proof. Algorithm 10 follows the same sampling framework used by Gillenwater et al. [16], which we recap here. The procedures starts with a partition $\{X_j\}_{j\in I}$ of the candidate set $\mathcal{S}$ such that all candidates in each X_j have the same utility, say u_j. Then, it samples $j \in I$ from the distribution $\mathbb{P}(j) \propto |X_j| \exp(\epsilon u_j/2)$, after which a candidate is uniformly chosen from X_j and returned as output. It has been shown that this procedure has the same output distribution as applying the exponential mechanism on $\mathcal{S}$ with utility u [16].

Algorithm 10 realizes the framework above by utilizing the utility function u^* and the partition scheme $\binom{\{0,\dots,d-1\}}{s} = \{\mathcal{C}\} \cup \{C_{i,j}\}_{1\le i\le m, m+1\le j\le d}$, where $\mathcal{C} = \{\sigma(\{1,\dots,m\} \cup P) \mid P \in \binom{\{m+1,\dots,d\}}{s-m}\}$ and $\mathcal{C}_{i,j} = \{\sigma(\{1,\dots,i-1\} \cup P \cup \{j\} \cup Q) \mid P \in \binom{\{i+1,\dots,j-1\}}{m-i-1}, Q \in \binom{\{j+1,\dots,d\}}{s-m}\}$. In line 3-9, the algorithm computes the size of $\mathcal{C}$ and $\mathcal{C}_{i,j}$, as well as the utilities of elements in each of them. In line 10, a particular partition set X is chosen using the formula $\mathbb{P}(X) \propto |X| \cdot \exp(\frac{\epsilon}{2} \cdot u_X)$, where u_X is the common utility of all candidates in X. After that, in line 11-17, a candidate is uniformly chosen from X and returned as output. Overall, Algorithm 10 realizes the aforementioned sampling framework, and therefore its output distribution is identical to applying the exponential mechanism on $\binom{\{0,\dots,d-1\}}{s}$ with utility u^*. Since u^* has sensitivity 1, the privacy of exponential mechanism ensures that Algorithm 10 is indeed ϵ-DP. □

6.4 Complete Protocol

We now describe the overall protocol which incorporates the local trimming technique described in the previous section. The protocol is named the *joint EM trimming algorithm*. It simply lets both parties locally compute two index sets I_1, I_2 respectively using Algorithm 10, and then execute the baseline protocol from Algorithm 4 on $I_1 \cup I_2$. We use ϵ, ϵ' to denote the privacy budget for the local trimming phase and the MPC selection phase respectively.

Algorithm 11 Joint EM trimming algorithm

Input: P_1, P_2 hold private vectors $x_1, x_2 \in \mathbb{N}^d$ respectively, $1 \le k \le m \le s \le d, \epsilon, \epsilon' > 0$

1: P_1 locally computes $I_1 \leftarrow$ top-m-in-$s(x_1, m, s, \epsilon)$ as in Algorithm 10
2: P_2 locally computes $I_2 \leftarrow$ top-m-in-$s(x_2, m, s, \epsilon)$ as in Algorithm 10
3: P_1, P_2 reveal I_1, I_2
4: P_1, P_2 run the baseline selection protocol in Algorithm 4 to compute the top-k indices on $(x_1 + x_2)_{(I_1 \cup I_2)}$ with privacy budget ϵ'

Since Algorithm 10 does not involve cryptographic operation, this step is insignificant in the runtime of the entire protocol, which is dominated by the two-party selection protocol in Algorithm 4. The array $(x_1 + x_2)_{(I_1 \cup I_2)}$ to selected on in the MPC phase has size at most $2s$. This reduction of input size to Algorithm 4 ensures an efficiency gain over the baseline protocol.

The privacy guarantee for Algorithm 11 follows from a simple application of the basic DP composition theorem.

Corollary 3. *Algorithm 11 is $(\epsilon + \epsilon')$-IND-CDP.*

Proof. From P_1's perspective, the result of running line 1-3 is ϵ-DP by Theorem 8, and the baseline protocol in line 4 satisfies ϵ'-IND-CDP by Corollary 2. The basic composition theorem then ensures that the entire protocol $(\epsilon + \epsilon')$-IND-CDP for P_2. A similar analysis holds for P_1. □

Chapter 7

Experimental results

In this chapter, we evaluate our proposed protocols on five public datasets, and compare their utilities and efficiencies. We start by describing the datasets, and then explain our choice of hyperparameters. Finally, we report the experiment results.

7.1 Datasets

We use five real-world datasets: Foods [21], Movies [20], Gowalla [7], BrightKite [7] and Foursquare [32]. The first two datasets, Foods and Movies, are based on review counts; while the other three are based on location check-ins. The two parties' datasets are generated by splitting each original dataset by the median of review/check-in timestamps. [1] The datasets are then processed such that each user contributes at most 1 to any vector entry. Table 7.1 shows dataset details.

7.2 Hyperparameter Selection

In this thesis, we have introduced three protocols for two-party DP selection: the baseline protocol (Algorithm 4), the threshold trimming algorithm (Algorithm 9) and the joint EM trimming algorithm (Algorithm 11). We now examine the hyperparameters used in these

[1] This data generation method is ad-hoc and we recognize the need for more realistic datasets for selection problem in multiparty settings.

Dataset	Dimension	Average	Median	Maximum	Correlation coefficient between two parties' vectors
Foods	166049	3.91	2	6340	0.62
Movies	59047	211.70	6	81491	0.66
Gowalla	1280969	1.62	2	2959	0.79
BrightKite	772967	0.72	1	3205	0.36
Foursquares	100191	1.70	1	1877	0.96

Table 7.1: Dataset details

algorithms. Since the baseline protocol contains no hyperparameter, we focus on the latter two.

The threshold trimming algorithm contains two hyperparameters: the maximum size m of each party's trimmed domain, and the ratio between ϵ and ϵ'. In order for the algorithm to select the true top-k indices in the sum vector, this true top-k index set should be contained in the union of two party's local noisy top-m index sets. In addition, m should be on a smaller magnitude than the entire domain size d, as the complexity of the MPC phase is dominated by $O(m \log m)$. We let $m = 30k$ in our experiments, which we consider enough to not miss out on any strong potential candidates in the local trimming stage.

Determining ϵ/ϵ' can be rephrased as deciding the ratio between ϵ' and the total privacy budget $k\epsilon + \epsilon'$. Recall that ϵ' is used to produce a geometric noise Geometric$(1 - e^{-\epsilon'})$, which is then added to the approximately $(m+1)$-th largest entry to produce the trimming threshold. Since the rest of the privacy budget $k\epsilon$ is used for the selection process, we would like ϵ' to be as small as possible, but not so small as to produce an overly high threshold that would rule out good potential candidates. We take ϵ' to be 5% of the total privacy budget. Even for the most stringent total privacy budget (0.4) used in the experiment, the resulting geometric noise on the threshold has a mean $1/(1 - e^{-\epsilon'})$ less than 60, which is much smaller than the $(m + 1)$-th largest entry in any of our dataset. This fact signifies that the resulting noisy threshold should still retain good potential candidates in the local trimming stage.

The joint EM trimming algorithm contains three hyperparameters: the size s of each party's trimmed domain, variable m used in the top-m-in-s problem in the local trimming step, and the ratio between ϵ and ϵ'. We set $s = 8192$ in all of our experiments. It is necessary for s to be on a higher magnitude than m to exhibit the advantage of the top-m-in-s relaxation, but also s should be notably smaller than d to ensure that the protocol

remains more efficient than the baseline protocol. As for privacy budget allocation, we simply set $\epsilon = \epsilon'$, as we cannot determine whether the local trimming step or the two-party selection step would require more privacy budget in practice. This leaves us with m. Recall that in the local trimming step in the joint EM trimming algorithm, every possible domain that contains the true top-m indices has the same optimal utility. Furthermore, for any candidate domain, the larger m is, the less utility it has in the exponential mechanism as realized by Algorithm 10 in the local trimming step. This fact hints that if the true top-k index set of the sum vector is already contained in the union of two party's local *true* top-m index sets, furthering increasing m would not produce better results. As such, m should be theoretically as small as possible such that the true top-k index set can still be contained in the union of two party's true local top-m index sets. In light of this conclusion, we set $m = 5k$ in the joint EM trimming algorithm in the following experiments. We do not test other values of m to avoid tailoring to the data.

7.3 Utility

We consider the following algorithms: the baseline protocol (BL), the threshold trimming algorithm (TT) and the joint EM trimming algorithm (JT). We also consider a straightforward strawman method (NA) in which the first party directly releases their local top-k indices in a differentially private way using Algorithm 1. We test every protocol on each dataset, with total privacy budget $\epsilon \in \{0.4, 0.6, 0.8, 1.0\}$, and $k \in \{10, 25, 50\}$. On each individual setting, we run 50 trials and report the average relative sum in Table 7.2. The relative sum is calculated as the sum of counts from chosen indices, divided by the sum of counts from the true top-k indices. We also report the standard deviation of the sum of counts from chosen indices in Table 7.3.

As expected, BL achieves the best result in almost all settings. However, there is no consistent winner in the rest of the algorithms. TT has a better general performance against JT and NA, whereas the latter two appear to offer competitive utilities. We deduce that despite our relaxation technique in JT's local trimming stage, publicly releasing both party's trimmed domain would still cost too much privacy budget for the algorithm to be practical, particularly for smaller datasets. For large datasets however, including Foods and Movies, both TT and JT are able to consistently beat the naive one-sided protocol NA. Furthermore, when $\epsilon \geq 0.6$ and $k \leq 25$, TT has close utility to BL in almost all scenarios. As for the standard deviation results, there is no observable trend suggesting that any algorithm has more stable performance than another.

		$\epsilon : 0.4$			0.6			0.8			1.0		
Dataset	Alg	$k : 10$	25	50	10	25	50	10	25	50	10	25	50
Foods	BL	1.00	1.00	0.75	1.00	1.00	0.92	1.00	1.00	0.92	1.00	1.00	1.00
	TT	1.00	1.00	0.68	1.00	1.00	0.85	1.00	1.00	0.98	1.00	1.00	0.99
	JT	1.00	0.95	0.55	1.00	1.00	0.73	1.00	1.00	0.79	1.00	1.00	0.89
	NA	0.95	0.90	0.53	0.95	0.92	0.65	0.95	0.92	0.79	0.95	0.92	0.87
Movies	BL	1.00	1.00	0.75	1.00	1.00	0.92	1.00	1.00	0.92	1.00	1.00	1.00
	TT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	JT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	NA	0.87	0.89	0.89	0.87	0.89	0.89	0.87	0.89	0.89	0.87	0.89	0.89
Gowalla	BL	1.00	0.96	0.43	1.00	1.00	0.63	1,00	1.00	0.80	1.00	1.00	0.93
	TT	1.00	0.85	0.30	1.00	0.99	0.50	1.00	1.00	0.66	1.00	1.00	0.80
	JT	1.00	0.69	0.37	1.00	0.76	0.48	1.00	0.78	0.51	1.00	0.79	0.53
	NA	0.99	0.69	0.23	0.99	0.91	0.42	0.99	0.96	0.52	0.99	0.97	0.65
BrightKite	BL	1.00	0.60	0.22	1.00	0.79	0.36	1.00	0.91	0.48	1.00	0.98	0.56
	TT	1.00	0.44	0.14	1.00	0.68	0.25	1.00	0.79	0.36	1.00	0.90	0.46
	JT	0.86	0.44	0.18	1.00	0.63	0.31	1.00	0.74	0.39	1.00	0.76	0.48
	NA	0.79	0.20	0.13	0.97	0.38	0.13	1.00	0.48	0.17	1.00	0.57	0.25
Foursquares	BL	1.00	0.66	0.33	1.00	0.85	0.41	1.00	0.97	0.51	1.00	0.99	0.60
	TT	1.00	0.61	0.31	1.00	0.79	0.39	1.00	0.95	0.49	1.00	0.99	0.57
	JT	0.92	0.50	0.18	1.00	0.62	0.33	1.00	0.72	0.38	1.00	0.78	0.44
	NA	1.00	0.51	0.20	1.00	0.66	0.33	1.00	0.77	0.38	1.00	0.90	0.46

Table 7.2: Utility results

		ϵ : 0.4			0.6			0.8			1.0		
Dataset	Alg	k : 10	25	50	10	25	50	10	25	50	10	25	50
Foods	BL	21.14	160.06	2023.81	0.00	53.48	1437.73	0.00	33.70	685.38	0.00	34.78	241.01
	TT	0.00	169.39	2388.67	0.00	79.34	1638.59	0.00	36.33	826.18	0.00	40.43	266.73
	JT	39.59	1372.40	2745.96	37.77	168.92	1593.85	0.00	143.92	1831.01	0.00	81.97	2131.70
	NA	72.31	820.46	2319.64	0.00	376.71	1436.75	51.66	326.70	1784.22	0.00	259.90	1128.47
Movies	BL	0.00	0.00	132.42	0.00	0.00	78.42	0.00	0.00	65.96	0.00	0.00	53.67
	TT	0.00	0.00	195.43	0.00	0.00	44.29	0.00	0.00	53.67	0.00	0.00	31.64
	JT	0.00	73.50	507.75	0.00	73.50	237.52	0.00	0.00	167.23	0.00	0.00	108.63
	NA	0.00	0.00	8644.85	0.00	0.00	7819.85	0.00	0.00	1310.40	0.00	0.00	6773.18
Gowalla	BL	12.39	719.41	1459.49	20.46	139.36	1509.64	7.87	54.90	1376.55	9.42	30.09	1115.94
	TT	29.37	1025.34	2303.45	11.60	225.73	1399.45	14.08	85.20	1468.21	7.87	70.12	1234.02
	JT	45.08	1201.33	1470.80	21.48	600.69	944.84	20.95	751.80	710.11	11.60	1121.76	1037.84
	NA	0.00	1121.52	2150.22	0.00	861.24	1124.38	0.00	297.46	1178.54	0.00	141.33	945.01
BrightKite	BL	34.04	637.25	820.56	12.13	578.41	981.21	10.10	446.94	553.56	8.55	297.49	614.96
	TT	26.59	712.21	593.30	24.15	759.57	600.01	7.68	584.69	931.73	9.42	526.63	816.27
	JT	882.37	804.85	1248.67	46.48	498.67	1042.38	17.49	394.78	775.59	18.87	201.32	687.66
	NA	445.62	650.91	27.78	459.52	484.75	532.37	136.23	753.21	838.89	0.00	730.50	892.83
Foursquares	BL	28.17	545.58	905.69	0.00	607.26	700.35	0.00	325.34	614.49	0.00	146.55	879.35
	TT	43.92	635.73	774.65	16.46	735.82	704.52	0.00	385.89	629.57	11.90	130.00	808.48
	JT	823.19	376.94	1330.17	35.51	748.46	918.00	20.19	545.19	544.77	11.62	502.49	733.87
	NA	88.32	452.58	1014.34	56.92	486.96	700.38	43.70	462.06	573.01	40.04	453.79	645.05

Table 7.3: Standard deviation results

7.4 Efficiency

We implemented and benchmarked all of our protocols using Obliv-C [34], an extension layer of C that compiles C program into Yao's garbled circuit protocols. For each dataset and each value of k, we empirically measure the gate complexity of the three proposed protocols. Numerical entries are all stored in 32-bit integers. Vector indices however, are allocated just enough bit-width instead of 32 bits, which makes swapping indices cheaper in the circuit.

Table 7.4 details the exact number of AND gates used by each protocol in each setting. We neglected reporting exact gate counts of BL when the performance gap between it and TT/JT can already seen to be huge. Note that the gate counts of TT and JT are primarily determined by the number of k, as their input size in the MPC phase is independent from the entire domain d. For these two protocols, when k is fixed, the only effect that the datasets have on their computation cost is the range of indices, which determines the indices' bit-width in the garbled circuit.

Both TT and JT uses 1-2 orders of magnitudes fewer gates than BL, due to the reduction in input size in the MPC phase. In addition, TT uses fewer gates than JT in all settings, although this efficiency gap can be seen to decrease as the number k increases. This is

Dataset	Alg	k : 10	25	50
Foods	BL	57,436,236	>57,436,236	>57,436,236
	TT	537,722	1,623,678	3,789,426
	JT	2,740,104	3,547,807	4,574,074
Movies	BL	20,106,313	>57,436,236	>57,436,236
	TT	527,432	1,591,122	3,711,056
	JT	2,740,104	3,547,807	4,574,074
Gowalla	BL	>20,106,313	>57,436,236	>57,436,236
	TT	553,157	1,672,512	3,906,587
	JT	2,740,104	3,547,807	4,574,074
BrightKite	BL	>20,106,313	>57,436,236	>57,436,236
	TT	548,012	1,656,234	3,867,796
	JT	2,740,104	3,547,807	4,574,074
Foursquares	BL	>20,106,313	>57,436,236	>57,436,236
	TT	532,577	1,607,400	3,750,241
	JT	2,740,104	3,547,807	4,574,074

Table 7.4: Gate counts

because increasing k will also increase the input domain size in TT, but this size is fixed for all k in JT.

7.5 Effects of Hyperparameters

We conclude this chapter by investigating the effects of m and s on TT and JT respectively. For TT, we repeat the previous experiment with $m \in \{5k, 10k, 30k\}$ and report the relative sums as a measure of utility. The result is in Table 7.5. We neglected dataset Movies in the result as the protocol achieves near-perfect utility in this dataset with all choices of m. Overall, increasing m also increases the utility of the protocol, as each party is less likely to miss out good potential candidates due to the increase in trimmed domain size. Note that this utility increase also comes at a cost of higher gate complexity in the MPC phase. We also repeat the experiment on JT with $s \in \{m, 8142\}$ and report the utility in Table 7.6. In all except a few cases, setting $s = 8142$ achieves a higher utility. We attribute this phenomenon to that the use of top-m-in-s relaxation lets each party's trimmed domain contain its true top indices more often when s is large. It is also suspected that changing s

would also change the optimal privacy budget allocation between the local trimming stage and the two-party selection stage. We leave this investigation to future work.

		ϵ : 0.4			0.6			0.8			1.0		
Dataset	Alg	k : 10	25	50	10	25	50	10	25	50	10	25	50
Foods	$m = 5k$	1.00	0.99	0.60	1.00	1.00	0.81	1.00	1.00	0.94	1.00	1.00	0.99
	$m = 15k$	1.00	0.99	0.64	1.00	1.00	0.85	1.00	1.00	0.98	1.00	1.00	0.99
	$m = 30k$	1.00	1.00	0.75	1.00	1.00	0.92	1.00	1.00	0.92	1.00	1.00	1.00
Gowalla	$m = 5k$	1.00	0.74	0.23	1.00	0.98	0.41	1.00	1.00	0.57	1.00	1.00	0.67
	$m = 15k$	1.00	0.79	0.24	1.00	0.99	0.46	1.00	1.00	0.59	1.00	1.00	0.73
	$m = 30k$	1.00	0.85	0.30	1.00	0.99	0.50	1.00	1.00	0.66	1.00	1.00	0.80
BrightKite	$m = 5k$	1.00	0.43	0.16	1.00	0.64	0.21	1.00	0.77	0.30	1.00	0.87	0.45
	$m = 15k$	1.00	0.41	0.14	1.00	0.64	0.22	1.00	0.79	0.33	1.00	0.88	0.42
	$m = 30k$	1.00	0.44	0.14	1.00	0.68	0.25	1.00	0.79	0.36	1.00	0.90	0.46
Foursquares	$m = 5k$	1.00	0.51	0.20	1.00	0.68	0.34	1.00	0.81	0.38	1.00	0.96	0.46
	$m = 15k$	1.00	0.54	0.24	1.00	0.68	0.36	1.00	0.87	0.40	1.00	0.99	0.49
	$m = 30k$	1.00	0.61	0.31	1.00	0.79	0.39	1.00	0.95	0.49	1.00	0.99	0.57

Table 7.5: Effect of m on utility of the threshold trimming algorithm

		ϵ : 0.4			0.6			0.8			1.0		
Dataset	Alg	k : 10	25	50	10	25	50	10	25	50	10	25	50
Foods	$s = m$	0.67	0.68	0.51	1.00	0.94	0.67	1.00	0.97	0.68	1.00	0.97	0.71
	$s = 8192$	1.00	0.95	0.55	1.00	1.00	0.73	1.00	1.00	0.79	1.00	1.00	0.89
Gowalla	$s = m$	0.45	0.30	0.25	0.68	0.42	0.44	1.00	0.71	0.49	1.00	0.74	0.52
	$s = 8192$	1.00	0.69	0.37	1.00	0.76	0.48	1.00	0.78	0.51	1.00	0.79	0.53
BrightKite	$s = m$	0.41	0.30	0.26	0.68	0.48	0.37	0.81	0.53	0.44	0.98	0.64	0.48
	$s = 8192$	0.86	0.44	0.18	1.00	0.63	0.31	1.00	0.74	0.39	1.00	0.76	0.48
Foursquares	$s = m$	0.77	0.50	0.35	0.77	0.50	0.35	0.77	0.50	0.36	0.77	0.51	0.43
	$s = 8192$	0.92	0.50	0.18	1.00	0.62	0.33	1.00	0.72	0.38	1.00	0.78	0.44

Table 7.6: Effect of s on utility of the joint EM trimming algorithm

Chapter 8

Conclusion

DP selection is a fundamental component in many DP data analysis tasks, and is far less well understood in the distributed model than in the central model. We propose two algorithms to solve DP selection in the two-party multi-computation setting, using Yao's garbled circuit as a building block. The advantage of our work lies in the sublinear complexity in the MPC phase, due to use of data-dependent heuristic. When the privacy budget is less restricted, our algorithms offer comparable empirical utility at a significantly lower computation cost than naively simulating report noisy max with distributed noise generation. We hope that our work would inspire more future research in DP selection in the distributed model, as well as other data-dependent DP algorithms.

References

[1] Aws Albarghouthi and Justin Hsu. Synthesizing coupling proofs of differential privacy. *Proc. ACM Program. Lang.*, 2(POPL), December 2017.

[2] V. E. Alekseev. Sorting algorithms with minimum memory. *Cybernetics*, 5(5):642–648, Sep 1969.

[3] Mitali Bafna and Jonathan Ullman. The price of selection in differential privacy. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 151–168. PMLR, 07–10 Jul 2017.

[4] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 307–314, New York, NY, USA, 1968. Association for Computing Machinery.

[5] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. Securely sampling biased coins with applications to differential privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 603–614, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 375–403, Cham, 2019. Springer International Publishing.

[7] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 1082–1090, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Seung Geol Choi, Dana Dachman-Soled, S. Dov Gordon, Linsheng Liu, and Arkady Yerukhimovich. Secure sampling with sublinear communication. Cryptology ePrint Archive, Paper 2022/660, 2022.

[9] Kelong Cong, Robin Geelen, Jiayi Kang, and Jeongeun Park. Revisiting oblivious top-k selection with applications to secure k-NN classification. Cryptology ePrint Archive, Paper 2023/852, 2023.

[10] Ivan Damgård, Hannah Keller, Boel Nelson, Claudio Orlandi, and Rasmus Pagh. Differentially private selection from secure distributed computing. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 1103–1114, New York, NY, USA, 2024. Association for Computing Machinery.

[11] Zeyu Ding, Daniel Kifer, Sayed M. Saghaian, Thomas Steinke, Yuxin Wang, Yingtai Xiao, and Danfeng Zhang. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise. *ArXiv*, abs/2105.07260, 2021.

[12] Zeyu Ding, Yuxin Wang, Danfeng Zhang, and Daniel Kifer. Free gap information from the differentially private sparse vector and noisy max mechanisms. *Proc. VLDB Endow.*, 13(3):293–306, November 2019.

[13] David Durfee and Ryan M. Rogers. Practical differentially private top-k selection with pay-what-you-get composition. *ArXiv*, abs/1905.04273, 2019.

[14] Cynthia Dwork and Kobbi Nissim. Privacy-preserving datamining on vertically partitioned databases. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 528–544, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[15] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014.

[16] Jennifer Gillenwater, Matthew Joseph, Andres Munoz, and Monica Ribero Diaz. A joint exponential mechanism for differentially private top-k. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7570–7582. PMLR, 17–23 Jul 2022.

[17] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *SIAM journal on computing (Print)*, 1989.

[18] Emil Julius Gumbel. *Statistical Theory of Extreme Values and Some Practical Applications : A Series of Lectures*. U.S. Government Printing Office, 1954.

[19] Wu Hao and Hanwen Zhang. Faster differentially private top-k selection: A joint exponential mechanism with pruning. *Advances in Neural Information Processing Systems 37*, 2024.

[20] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[21] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 507–517, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[22] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1389–1406, New York, NY, USA, 2017. Association for Computing Machinery.

[23] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, page 486–498, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *2008 IEEE 24th International Conference on Data Engineering*, pages 277–286, 2008.

[25] Ryan McKenna and Daniel Sheldon. Permute-and-flip: a new mechanism for differentially private selection. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[26] Andrés Muñoz Medina and Jennifer Gillenwater. Duff: A dataset-distance-based utility function family for the exponential mechanism. *ArXiv*, abs/2010.04235, 2020.

[27] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 650–661, New York, NY, USA, 2012. Association for Computing Machinery.

[28] Ricardo Silva Carvalho, Ke Wang, Lovedeep Gondara, and Chunyan Miao. Differentially private top-k selection via stability on unknown domain. In Jonas Peters and David Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 1109–1118. PMLR, 03–06 Aug 2020.

[29] Thomas Steinke. Multi-central differential privacy, 2020.

[30] Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, January 1968.

[31] Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. Proving differential privacy with shadow execution. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 655–669, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Dingqi Yang, Daqing Zhang, Vincent W. Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142, 2015.

[33] Andrew Chi-Chih Yao. Bounds on selection networks. *SIAM J. Comput.*, 9(3):566–582, August 1980.

[34] Samee Zahur and David Evans. Obliv-c: A language for extensible data-oblivious computation. *IACR Cryptol. ePrint Arch.*, 2015:1153, 2015.

[35] Yuqing Zhu and Yu-Xiang Wang. Adaptive private-k-selection with adaptive k and application to multi-label pate. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 5622–5635. PMLR, 28–30 Mar 2022.