Secure Multi-party Computation of Differentially Private Heavy Hitters

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

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

ABSTRACT

Private learning of top-k, i.e., the k most frequent values also called *heavy hitters*, is a common industry scenario: Companies want to privately learn, e.g., frequently typed new words to improve suggestions on mobile devices, often used browser settings, telemetry data of frequent crashes, heavily shared articles, etc.

Real-world deployments often use local differential privacy, where distributed users share locally randomized data with an untrusted server. Central differential privacy, on the other hand, assumes access to the raw data and applies the randomization only once, on the aggregated result. These solutions either require large amounts of users for high accuracy (local model) or a trusted third party (central model). We present multi-party computation protocols HH and PEM of sketches (succinct data structures) to efficiently compute differentially private top-k: HH has running time linear in the size of the data and is applicable for very small data sets (hundreds of values), and PEM is sublinear in the data domain and provides better accuracy than HH for large data sizes. Our approaches are efficient (practical running time, requiring no output reconstruction as other sketches) and more accurate than local differential privacy even for a small number of users. In our experiments, we were able to securely compute differentially private top-k in less than 11 minutes using 3 semi-honest computation parties distributed over the Internet with inputs from hundreds of users (HH) and input size that is independent of the user count (PEM, excluding count aggregation).

CCS CONCEPTS

• **Security and privacy** → **Data anonymization and sanitization**; **Privacy protections**; **Privacy-preserving protocols**.

KEYWORDS

Heavy Hitters, Differential Privacy, Secure Multi-party Computation, Sketches

ACM Reference Format:
Jonas Böhler and Florian Kerschbaum. 2021. Secure Multi-party Computation of Differentially Private Heavy Hitters. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3460120.3484557

1 INTRODUCTION

The goal of federated top-k discovery is to learn the k most frequent values, also called *heavy hitters*, in a distributed data set. *Differential privacy* (DP) [34, 36], a rigorous privacy notion, restricting what can be inferred about any individual in the data, is widely deployed to mitigate privacy risks and regulatory concerns, when the data is generated by users. E.g., Apple deploys DP to privately learn frequently typed words on mobile devices to improve auto-complete suggestions and to detect websites with large resource consumptions to optimize the browsing experience in iOS and macOS [8, 74]. Google privately detects popular Chrome browser settings [40, 42] as well as busy times for businesses in Google Maps [17]. Also, Microsoft deploys differentially private telemetry data collection in Windows 10 (Creators Fall Update) across millions of devices [30] and LinkedIn's Audience Engagement API lets marketers query, e.g., DP top-k shared articles among users with a specific skill set [70, 71]. Real-world deployments [8, 17, 30, 40, 42] mainly implement the *local model* of DP, i.e., users locally randomize their data and send it to an untrusted aggregator. In the *central model*, e.g., used by LinkedIn [70, 71] and the US Census bureau [1], a trusted party has access to the raw data, which only needs to apply randomization once, on the aggregated result. The local model has fewer assumptions (no trusted party) but generally requires up to exponentially more samples to achieve the optimal accuracy offered by the central model at the same privacy parameter [52].

We present a novel alternative with central model accuracy, that requires no trusted party, and is efficiently computable. Our protocol provides high accuracy even for a small number of users, which is the most challenging regime for DP [16, 18, 66], via efficient *secure multi-party computation* (MPC), which enables parties to jointly compute a function without revealing their inputs [45]. The straightforward algorithms to accurately detect heavy hitters are inefficient in MPC [61], and hence we employ *sketches*, clever approximate algorithms with succinct data representation, for streams [27] or large domains [78] to make the secure multi-party computation efficient. Typically, sketches store counters indexed by multiple hash functions, e.g., count-min sketch [27, 61] or Bloom filters [40, 42], and local-model users apply domain reduction (e.g., hashing) [11, 12, 78]. However, hash-based techniques require costly reconstruction, e.g., iterating/hashing the entire domain to find heavy hitters, whereas our sketches directly store heavy hitters or their bit prefixes. Our key insight is that adapting suitable sketches (without reconstruction) enables efficient MPC of DP heavy hitters with high accuracy. We present two MPC protocols to discover the differentially private top-k on distributed user data: HH and PEM. Our protocols are based on state-of-the-art solutions for heavy hitter detection – HH is build upon *non-private* detection in data streams [27], and PEM adapts the *local DP* method from [78] – realized as efficient MPC implementations of *central DP* randomizations [32, 33, 79]. HH has running time linear in the size of the data and is applicable for very small data sets (hundreds of values). PEM is sublinear in the data domain (i.e., linear in the bit-length of domain size) and provides better accuracy than HH for a large number of users (thousands to millions). We improve upon related work for MPC of DP heavy hitters [20, 61, 65], which is linear in the size of the data, and requires reconstruction or is not optimized for DP resulting in lower accuracy (see Section 6 for details). In summary, our contributions are DP heavy hitter MPC protocols

- with high accuracy even for small data sets or few users (Section 5),
- that are efficient (practical running times, reconstruction-free) (Section 5),
- secure in the semi-honest model and extendable to malicious adversaries (Section 4.5),
- both implemented and evaluated in two MPC frameworks, SCALE-MAMBA [4] and MP-SPDZ [53], using 3 semi-honest computation parties in a WAN with 100 ms network delay, 100 Mbits/s bandwidth achieving running times of less than 11 minutes (Section 5).

In Section 2, we describe preliminaries. In Section 3, we describe an ideal version our protocol with a trusted third party, which we later replace with MPC. We present our secure multi-party computation of DP heavy hitters in Section 4. We provide a detailed evaluation in Section 5, describe related work in Section 6 and conclude in Section 7.

2 PRELIMINARIES

Next, we detail the problem, and introduce preliminaries for secure multi-party computation and differential privacy.

Problem Description: We consider a set of n input parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where party P_i holds a datum d_i, and $D = \{d_1, \ldots, d_n\} \in U^n$ denotes the combined data set with underlying data domain U. We want to find heavy hitters, more formally:

Definition 1 (Top-k or Heavy Hitter). *Datum $d \in D$ is a* top-k element *if its frequency f_d in D is among the k most frequent elements, where $f_d = |\{x \mid x \in D \text{ and } x = d\}|/|D|$.*

We release *at most k* heavy hitters like Durfee and Rogers [32], as thresholding might drop small noisy counts (unlikely to be heavy hitters). Durfee and Rogers note that for flat histograms, i.e., all counts are similar, thresholding outputs nothing instead of (almost) uniformly random elements, which provides more data insights (i.e., flat histogram) than randomness.

We define accuracy like Wang et al. [78] as the *normalized cumulative rank* (NCR). Unlike the F1 score, NCR also considers an element's frequency:

Definition 2 (Normalized Cumulative Rank (NCR)). *Let C_k denote the* actual *top-k values and C the* presumed *top-k (returned by* HH, PEM*). The* normalized cumulative rank *of C is $\sum_{c \in C} r(c)/\sum_{c' \in C_k} r(c')$, where rank $r(c_i) = k + 1 - i$ for i-th most frequent element $c_i \in C_k$ and zero otherwise.*

Basically, detecting the most frequent element increases the cumulative rank by k, the second most frequent element adds another $k - 1$, etc., and the sum is normalized to $[0, 1]$ by dividing it with maximum score $\sum_{c' \in C_k} r(c') = k(k + 1)/2$.

We aim for efficient MPC of DP heavy hitters using sketches for approximate counts. Straightforward, exact MPC requires a counter per domain element, i.e., for n data values from domain of size u the running time complexity is in $O(un)$. Related work for MPC either requires costly reconstruction linear in u, or is not optimized for DP and does not provide high accuracy for small data sets (see Section 6). Our protocols HH, resp. PEM, enable $O(nt)$, resp. $O(gc \log(c))$, for small parameters $t \in O(1), c = O_\eta(k), g < \log(u)$ (Section 4.3), and the communication complexity for input parties is in the order of kilobytes (Section 5).

Secure Multi-party Computation: Secure multi-party computation (MPC) [45] allows a set of three or more parties to jointly compute a function $y = f(d_1, \ldots, d_n)$ while protecting their inputs d_i. The computation must be *correct*, i.e., the correct y is computed, and *secret*, i.e., only y and nothing else is revealed. There are two main implementation paradigms for MPC [41, 54]: *garbled circuits* [46, 57, 80], where the parties construct a (large, encrypted) circuit and evaluate it at once, and *secret sharing* [15, 29, 67, 72], where the parties interact for each circuit gate. In general, the former allows for constant number of rounds but requires larger bandwidth (as fewer, but bigger messages are sent), and the latter has low bandwidth (small messages per gate) and high throughput, where the number of rounds depends on the circuit depth. We will focus on secret-sharing-based MPC as our goal is an efficient implementation in a network with reasonable latency. Informally, a (t, n)-secret sharing scheme splits a secret s into n shares s_i and at least t shares are required to reconstruct the secret. We use $\langle s \rangle = (s_1, \ldots, s_n)$ to denote the sharing of s among n parties (for a formal definition see, e.g., Evans et al. [41]). Recent works, e.g., SCALE-MAMBA [4], BDOZ [15], SPDZ [29], improve MPC performance by combining a computationally secure *offline phase*, to exchange correlated randomness (e.g., Beaver triples [13]), with an information-theoretic secure *online phase*. The latter is generally more efficient since the former requires asymmetric cryptography [55]. MPC can be implemented in two models with different trust assumptions: in the *semi-honest model* (passive) adversaries do not deviate from the protocol but gather everything created during the run of the protocol, in the *malicious model* (active) adversaries can deviate from the protocol (e.g., alter messages). We consider n *input parties* with sensitive input, and $m = 3$ semi-honest *computation parties*, i.e., non-colluding untrusted servers. The input parties create and send shares of their input to the computation parties, which run the secure computation on their behalf. We assume semi-honest parties, discuss extensions of our protocol to malicious parties, and implement our protocol with two frameworks, SCALE-MAMBA [4] and MP-SPDZ [53], and compare them.

Differential Privacy: Differential privacy (DP), introduced by Dwork et al. [34, 36], is a strong privacy guarantee restricting what a mechanism operating on a sensitive data set can output. Informally, when the input data set changes in a single element, the effect on the output is bounded. The formal definition is as follows:

Definition 3 (Differential Privacy). *A mechanism $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy, where $\epsilon, \delta \geq 0$, if for all neighboring data sets $D \simeq D'$, i.e., data sets differing in a single entry, and all sets*

$S \subseteq Range(\mathcal{M})$

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

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

$(\epsilon, 0)$-DP is also called *pure DP*, whereas *approximate DP* allows an additional, additive privacy loss $\delta > 0$. Typically, δ is negligible in the size of the data [37]. While we present pure DP mechanisms in the following our protocols apply them in combination with δ-based thresholds from [32, 79], thus, our protocols satisfy approximate DP. The DP definition can be adapted to fit different deployment models (which we compare in Section 2) and next we describe two variations: local and computational DP. Note that Definition 3 assumes that mechanism $\mathcal{M}$ has access to the raw data D. In a *distributed* setting, where each user locally randomizes her value v, the notion of *local* DP (LDP) [52] is used. Here, the output changes for any input $v, v' \in D$ are ϵ-bounded, i.e., $\Pr[\mathcal{M}(v) \in S] \leq \exp(\epsilon) \cdot \Pr[\mathcal{M}(v') \in S]$. Definition 3 holds against an unbounded adversary, however, we consider computationally-bounded, semi-honest parties for a joint secure computation realized with (t, m)-secret sharing. We use the definition from Eigner et al. [39], where $\mathtt{VIEW}^{p}_{\Pi}(D)$ is the view of party p during the execution of protocol Π on input D, including all exchanged messages and internal state, and κ is a security parameter:

Definition 4 (Distributed Differential Privacy). *A randomized protocol Π implemented among m computation parties $\mathcal{P} = \{P_1, \ldots, P_m\}$, satisfies* distributed differential privacy *w.r.t. a coalition $C \subset \mathcal{P}$ of semi-honest computation parties of size t, if the following condition holds: for any neighbors D, D' and any possible set S of views for protocol Π,*

$$Pr\Big[\mathtt{VIEW}^{C}_{\Pi}(D) \in S\Big] \leq \exp(\epsilon) \cdot Pr\Big[\mathtt{VIEW}^{C}_{\Pi}(D') \in S\Big] + \delta_\kappa,$$

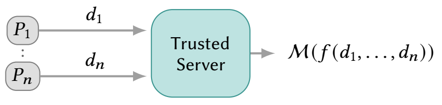
where δ_κ is negligible in the security parameter κ.

The definition covers a malicious setting by replacing the semi-honest parties $\mathcal{P}$ and semi-honestly secure protocol Π with malicious parties and a maliciously secure protocol.

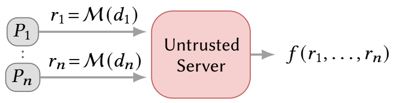
Noise added to the function output is one way to achieve differential privacy, e.g., via the *Laplace mechanism $\mathcal{M}_{\mathcal{L}}$* [37]:

Definition 5 (Laplace Mechanism $\mathcal{M}_{\mathcal{L}}$). *The* Laplace mechanism *$\mathcal{M}_{\mathcal{L}}$, for function $f : U^n \rightarrow \mathbb{R}$ which has sensitivity $\Delta f = \max_{\forall D \simeq D'} |f(D) - f(D')|$, releases $f(D) + \mathrm{Laplace}(\Delta f / \epsilon)$, where $\mathrm{Laplace}(b)$ denotes a random variable from the Laplace distribution with scale b and density $\mathrm{Laplace}(x; b) = \frac{1}{2b} \exp\Big(-\frac{|x|}{b}\Big)$.*

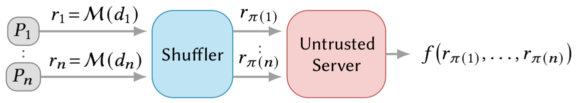
An alternative is probabilistic output selection via the *exponential mechanism $\mathcal{M}_{\mathcal{E}}$*, introduced by McSherry and Talwar [60]. The exponential mechanism expands the application of differential privacy to functions with non-numerical output, or when the output is not robust to additive noise [56]. The mechanism is exponentially more likely to select "good" results where "good" is quantified via a utility function $u(D, r)$ which takes as input a database $D \in U^n$, and a potential output $r \in \mathcal{R}$ from a fixed set of arbitrary outputs $\mathcal{R}$. Informally, $\mathcal{M}_{\mathcal{E}}$ selects an output r with probability proportional to $\exp\Big(\frac{\epsilon u(D,r)}{2\Delta u}\Big)$. Recently, Durfee and Rogers [32] showed that the argmax over utility scores with additive noise from the Gumbel distribution is equivalent to $\mathcal{M}_{\mathcal{E}}$[1] [32, Lemma 4.2], and we call this the *Gumbel mechanism* (defined in Appendix A).



(a) Central Model



(b) Local Model

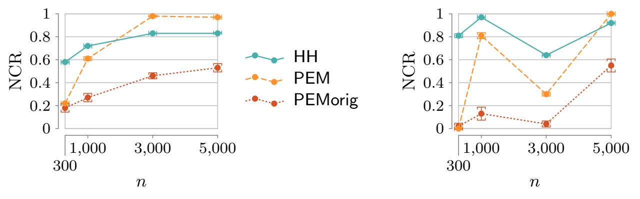


(c) Shuffle Model with permutation π

Figure 1: Implementation models for DP mechanism $\mathcal{M}$. Party P_i sends a message (raw d_i or randomized r_i) to a server, who evaluates function f on the combined messages.

DP Implementation Models: Different implementation models for DP exist, shown in Figure 1, and MPC combines their respective benefits [18]. The *central model* (Figure 1a) assumes a trusted server receives raw data from each client. The server applies DP mechanism $\mathcal{M}$ on the raw data, which achieves optimal accuracy. The *local model* [52] (Figure 1b) assumes an untrusted server and clients locally apply $\mathcal{M}$ on their data before sending it to the server. As the randomization is applied multiple times (at each client), the accuracy is limited. Hence, it requires a very large number of users to achieve accuracy comparable to the central model [16, 25, 50, 52, 59]. Specifically, an exponential separation between local and central model for accuracy and sample complexity was shown [52]. Recently, an intermediate *shuffle model* (Figure 1c) was introduced [16, 25]: A trusted party is added between client and server in the local model, the shuffler, who does not collude with anyone. The shuffler permutes and forwards the randomized client values. The permutation breaks the mapping between a client and her value, which reduces randomization requirements. While the shuffle model is more accurate than the local model (especially augmented with secure summation [10, 44]), we also use thresholding to satisfy DP (requiring secure comparisons). The shuffle model (or mix-nets in general) requires k iterations to find k heavy hitters [20, Section 2.4] or costly sketch reconstruction (as the local model [40, 42, 78]), whereas thresholding allows more accurate and efficient one-shot discovery [32, 33]. As our goal is high accuracy without trusted parties even for small number of users, we *simulate* the central model in a distributed setting via MPC as commonly found in DP literature [18, 19, 35, 39, 47, 68, 69, 73]. General MPC suffers from overhead for computation as well as communication. However, MPC combines the respective benefits of the models, namely, high accuracy without disclosing values to a third party,

[1]Similar to *Report One-Sided Noisy Arg Max* [37, Section 3.4], which uses the Exponential distribution.



(a) Zipf **(b) Retail prices [75]**

Figure 2: Accuracy (NCR) of our MPC protocols PEM **and** HH **compared to LDP protocol** PEMorig **[78] for parameters $k = 8$, $|U| = 2^{32}$, $\epsilon = 2$ with $n \in \{300, 1000, 3000, 5000\}$.**

and we present two *efficient* MPC protocols for DP top-k: HH has running time linear in the size of the data and is applicable for very small data sets (hundreds of values). PEM is sublinear in the data domain (linear in the bit-length of the data domain) and provides better accuracy than HH for larger data sizes.

We measure top-k accuracy like Wang et al. [78] via non-cumulative rank (NCR), which is similar to F1 score weighted by an element's rank, where the most frequent value has rank k, the second most frequent rank $k - 1$, etc. (Definition 2). Figure 2 illustrates that our protocols provide higher accuracy than the state-of-the-art LDP heavy hitter approach by Wang et al. [78], which we denote PEMorig and detail in Section 3.2.1. We used synthetic data from the same Zipf distribution as Wang et al. [78][2] as well as a real-world Online retail data set [75] and provide more detailed evaluation in Section 5. Note that the local model and hash-bashed sketches (e.g., invertible/counting Bloom filters, count-min sketch) can require the aggregator to consume significant computational resources to reconstruct estimates from perturbed reports: PEMorig performs $n2^q$ hash computations to match potential heavy hitters with reported (randomized) hashes. Even for small data of size $n = 1000$ with recommended $q = 20$, around 1 billion hashes are computed. Likewise, RAPPOR [42, 43] (follow-up to [40]) detects frequent strings (e.g., browser homepage, installed software) by estimating joint probabilities of (perturbed) n-grams via the expectation maximization algorithm, with complexity $O(|D||L|^{nr})$ for r reported n-grams per party for string alphabet L [43, Section V.B].

Our MPC protocols have better running time complexity than the above mentioned approaches (Section 4.3), provide accuracy as in the central model (Section 5), and the computation can be outsourced to a few computation parties independent of the number of users (Section 4.5). Hence, we show that the adaptation of *central DP sketches suitable for heavy hitters* (i.e., without reconstruction) is more accurate than LDP and efficient enough for MPC.

3 FEDERATED HEAVY HITTERS

The following ideal functionalities $\mathcal{F}_{\text{HH}}$ and $\mathcal{F}_{\text{PEM}}$ describe our protocols as executed by a trusted third party, which we later replace by implementing them with optimized MPC protocols as HH and PEM, respectively. The straight-forward algorithms to accurately detect heavy hitters are inefficient in MPC. Therefore, we employ

[2]Zipf(1.5), i.e., the j-th most frequent value appears with probability proportional to $1/j^{1.5}$.

```
(1) Define map $T$ of size $t$ to associate a datum with a count.
(2) For each user reported datum $d \in D$:
    (a) If $d \in T$, then increment counter $T[d]$.
    (b) Else if $|T| < t$, then add $d$ to $T$, and set $T[d]$ to 1.
    (c) Else, decrement all counters $T[i]$, remove $i$ from $T$ if $T[i] = 0$.
(3) For each item $i \in T$:
    (a) Add noise Laplace$(\Delta/\epsilon)$ to count $T[i]$.
    (b) Remove $i$ from $T$ unless
            $T[i] \geq \tau_{\text{HH}}$,
        where $\tau_{\text{HH}} = 1 - \Delta \log\left(2 - 2(1-\delta)^{1/\Delta}\right)/\epsilon$.
(4) Output items in $T$ sorted by their noisy count.
```

Figure 3: Ideal functionality $\mathcal{F}_{\text{HH}}$ combines heavy hitter detection in streams [27, Alg. 1] with DP-bounded count release [79, Th. 2].

sketches, i.e., clever approximate algorithms that work over *streams with unknown domains* [27] (non-private) or *support large domains* [78] (local model) to make the secure multi-party computation efficient. The employed sketches do not require hashing or domain reduction (e.g., Bloom filters [40], or matrix projection [12]) and avoid the reconstruction effort and information loss associated with it [78]. Clients only send a single message – either their value ($\mathcal{F}_{\text{HH}}$) or a bit vector indicating the bit-prefix of their value ($\mathcal{F}_{\text{PEM}}$) – and the server updates a map that associates client messages with a count. We utilize central model thresholds [32, 33, 79] and show that $\mathcal{F}_{\text{HH}}$, $\mathcal{F}_{\text{PEM}}$ are differentially private. In the following, we let Δ denote the maximum number of counts an individual can influence, e.g., $\Delta = 1$ if we query countries of origin, or $\Delta \geq 1$ for current and former employers.

3.1 Ideal Functionality $\mathcal{F}_{\text{HH}}$

Cormode and Hadjieleftheriou [27] surveyed algorithms for (non-private) heavy hitter detection in data streams and found counter-based approaches, to be the best w.r.t. accuracy, speed and space (which was re-confirmed by more recent work [7]). Next we describe a non-private counter-based approach, which we augment to be privacy-preserving.

3.1.1 Non-private Misra-Gries. The counter-based approach *Misra-Gries* [63],[27, Alg. 1], makes up all steps of ideal functionality $\mathcal{F}_{\text{HH}}$ in Figure 3, excluding the DP thresholding in step 3. Misra-Gries uses counters to track the frequency of already seen elements in a data stream and provides the following guarantee [7, Lemma 1]:

Lemma 1. *Misra-Gries run on D of size n with t counters provides a frequency estimate $\widehat{f_d}$ for all $d \in D$ satisfying $0 \leq f_d - \widehat{f_d} \leq n/(t+1)$.*

Recent improvements, e.g., [7], reduce the expected number of times the expensive decrement branch is executed (2c in Figure 3), as it requires updating the entire map T. However, as we later implement $\mathcal{F}_{\text{HH}}$ with MPC, which must hide the control flow to prevent leakage, we cannot apply them and focus on the original version. Note that $\mathcal{F}_{\text{HH}}$, due to its use of Misra-Gries, does not require any domain knowledge or distribution assumptions. Also, if the map size is equal to the size of the (small) data set, $\mathcal{F}_{\text{HH}}$ computes an *exact* histogram over an unknown data domain.

3.1.2 *Differentially private $\mathcal{F}_{\mathsf{HH}}$.*

The ideal functionality $\mathcal{F}_{\mathsf{HH}}$ in Figure 3 approximates counts for frequent values seen so far via [27, Alg. 1] but only releases noisy counts that exceed the δ-based threshold τ_{HH} defined by Wilson et al. [79, Th. 2].

THEOREM 1. *$\mathcal{F}_{\mathsf{HH}}$ provides $(\Delta\epsilon, \delta)$-differential privacy.*

PROOF. Wilson et al. [79, Th. 2] proof that the threshold τ_{HH} satisfies $(\Delta\epsilon, \delta)$-DP for counts of unique user contributions in SQL. I.e., non-empty groups with noisy counts of say column 1 grouped by column 2 are released if they exceed the threshold, and the threshold bounds the probability for releasing differing results between neighbors. We briefly sketch their proof: A noisy count will be at least τ with probability $p = \frac{1}{2}e^{-\frac{\epsilon(\tau-1)}{\Delta}}$ (property of Laplace distribution). The probability for bad events (e.g., releasing a count for a data set but not its neighbor) is bounded as $p^{\Delta} \leq \delta$ and solving for τ provides τ_{HH}. As we assume a single value per user, each count qualifies as a unique contribution per user, allowing us to use the same threshold τ_{HH}. □

3.2 Ideal Functionality $\mathcal{F}_{\mathsf{PEM}}$

Wang et al. [78] present a "prefix extension method" (PEM) for LDP heavy hitter detection and show that it provides higher accuracy than other LDP approaches [11, 12, 42]. We adapt their local model protocol, which we denote PEMorig, for our central model protocol $\mathcal{F}_{\mathsf{PEM}}$, and describe them next.

3.2.1 *Local model* PEMorig.

We briefly describe PEMorig and refer to Appendix B for details. PEMorig leverages overlapping segments by iteratively finding frequent prefixes of increasing lengths and clients are split evenly in disjoint groups. The first group reports perturbed $(\gamma + \eta)$-bit prefixes of their datum to a server, and the server estimates the frequencies of all prefix candidates (i.e., all binary strings with the same length as the bit prefix). Then, the prefix length is extended by η, the second group reports their perturbed prefixes of length $\gamma + 2\eta$, and the server estimates frequencies of prefixes that extend the top-2^{γ} prefixes of the previous group. This is repeated until the prefix length reaches the domain bit-length b. To create the LDP reports, a client first reduces the domain size to $u = \lceil \exp(\epsilon) + 1 \rceil$ via *optimal local hashing* [77], then applies *generalized randomized response* on the value from the reduced domain (which returns the input with probability $\frac{\exp(\epsilon)}{\exp(\epsilon)+u-1}$ and any other domain value with equal probability). To estimate candidate frequency, the server hashes all current prefix candidates, and matches them to each report. The parameter η provides the following trade-off: Smaller values lead to more groups but less (hash) computations, whereas larger values produces fewer groups but requires more computational resources. Note that more groups means fewer counts per prefix candidate which can lead to reduced accuracy. Wang et al. [78] set $\gamma = \lceil \log_2 k \rceil$ and limit the number of hash computations per report to 2^{20} (i.e., set η to the largest integer satisfying $g2^{\gamma+\eta} < 2^{20}$ for $g = \lceil (b - \gamma)/\eta \rceil$ groups). Overall, PEMorig, with these parameters, requires the server to compute $n2^{20}$ hashes.

3.2.2 *Central model $\mathcal{F}_{\mathsf{PEM}}$.*

Our protocol $\mathcal{F}_{\mathsf{PEM}}$, shown in Figure 4, also leverages extending prefixes to find heavy hitters over distributed data. Unlike PEMorig, $\mathcal{F}_{\mathsf{PEM}}$ operates on actual counts instead of estimates from perturbed reports to increase the accuracy. We later implement $\mathcal{F}_{\mathsf{PEM}}$ with MPC to protect the counts and use $\eta \in \{4, 5\}$ in our evaluation as this provides a practical trade-off between computational efficiency and accuracy for a small number of users (Section 5). $\mathcal{F}_{\mathsf{PEM}}$ releases *intermediate* results (set of frequent prefix candidates) to improve the frequency estimation in multiple rounds, unlike minimal functionality HH, which only releases the *final* result. However, this does not violate privacy: Differential privacy is enforced in line 2(d)ii of Figure 4 by only releasing values whose associated (noisy) frequencies exceed a threshold. The privacy budget of $\mathcal{F}_{\mathsf{PEM}}$ is given next.

```
(1) Initialize prefix set $S = \{0,1\}^{\lceil \log k \rceil}$, and split user data $D$
    in $g = \left\lceil \frac{b - \lceil \log k \rceil}{\eta} \right\rceil$ disjoint groups $D_1, \ldots, D_g$.
(2) For each group $j \in \{1, \ldots, g\}$:
    (a) Initialize empty map $T$ to associate a prefix with a count, and
        candidate prefix set $C = S \times \{0,1\}^{\eta}$.
    (b) For each prefix $c \in C$:
        (i) Set $T[c] = \sum_{d \in D_j} \zeta_d^c$, where $\zeta_d^c \in \{0,1\}$ is a user report
            indicating if her value $d$ matches prefix candidate $c$.
    (c) Set $S = \{\}$ and $z = \min_{c \in C} T[c] + \text{Laplace}(1/\epsilon)$.
    (d) For the top-$k$ prefixes $c_k \in C$:
        (i) Add noise Laplace$(1/\epsilon)$ to count $T[c_k]$.
        (ii) Add $c_k$ to $S$ if
                 $T[c_k] \geq \tau_{\mathsf{PEM}} + z$,
             where $\tau_{\mathsf{PEM}} = 1 + \log(\Delta/\delta)/\epsilon$.
    (e) Output items in $S$ sorted by their noisy count.
```

Figure 4: Ideal functionality $\mathcal{F}_{\mathsf{PEM}}$ combines distributed heavy hitter detection [78] with central-DP top-k [32, 33].

THEOREM 2. *$\mathcal{F}_{\mathsf{PEM}}$ provides $(\Delta\epsilon, \frac{\delta}{4}(\exp(\Delta\epsilon) + 1)(3 + \log(\Delta/\delta)))$-differential privacy.*

PROOF. The claim holds for a single group, i.e., step 2d, as the thresholding satisfies $\left(\Delta\epsilon, \frac{\delta}{4}(\exp(\Delta\epsilon) + 1)(3 + \log(\Delta/\delta))\right)$-DP [32, Lemma 6.1]. Now we expand this, without additional privacy cost, to all groups (step 2): Recall, we compute counts on *disjoint* subsets (i.e., D_g of D for group g). Thus, we never count a user contribution more than once. Parallel composition [56, Section 2.2.2] applies *maximum* (instead of the sum) over the privacy budget for all steps as total budget. As we use the *same* budget per step the maximum is equal to that of a single step, which concludes the proof. □

3.2.3 *Unrestricted Sensitivity.*

In the case of unrestricted sensitivity, i.e., Δ much larger than $|C|$, Durfee and Rogers [32] use Gumbel noise instead of Laplace noise. With Gumbel noise $\mathcal{F}_{\mathsf{PEM}}$ is $(\approx \sqrt{k}\epsilon, \delta)$-DP [32, Th. 1] (i.e., with a dependence on k instead of Δ). To support unrestricted sensitivity in $\mathcal{F}_{\mathsf{PEM}}$ the Laplace noise in lines 2c, 2(d)i of Figure 4 changes to Gumbel noise with the same scale with new threshold $\tau_{\mathsf{PEM}} = 1 + \log(|C|/\delta)/\epsilon$ (i.e., Δ replaced by $|C|$). In the following, we focus on the setting with restricted sensitivity, i.e., Laplace noise, but our protocol can be extended to the unrestricted case by using distributed Gumbel noise (Section 3.4).

3.3 When to use $\mathcal{F}_{\text{HH}}$ or $\mathcal{F}_{\text{PEM}}$

Theorem 3. *For data set D of size n, $\mathcal{F}_{\text{HH}}$ with fixed map size t provides better accuracy in expectation than $\mathcal{F}_{\text{PEM}}$ if*

$$\tau_{\text{HH}} + \frac{n}{t+1} < f_{k\text{-}th} \le g \cdot \left(\tau_{\text{PEM}} + f_{|C|\text{-}th}\right)$$

where $f_{k\text{-}th}$ is the frequency of the k-th most frequent element in D, g is the number of groups in $\mathcal{F}_{\text{PEM}}$, $|C|$ is the size of the candidate set in $\mathcal{F}_{\text{PEM}}$, and $\tau_{\text{HH}}, \tau_{\text{PEM}}$ as in Figures 3, 4 respectively.

Proof. We consider the cases where $\mathcal{F}_{\text{HH}}$ releases a candidate and $\mathcal{F}_{\text{PEM}}$ does not. $\mathcal{F}_{\text{HH}}$ releases candidate c if $T[c]+\text{Laplace}(1/\epsilon) > \tau_{\text{HH}}$. $\mathcal{F}_{\text{HH}}$ uses estimated frequency $T[c] = \widehat{f_c}$, which is at most $n/(t+1)$ below actual frequency f_c (Lemma 1). Thus, $T[c] > f_c - n/(t+1)$ using the fact that Laplace noise is 0 in expectation and replacing f_c with $f_{k\text{-th}}$, we have $\tau_{\text{HH}} + n/(t+1) < f_{k\text{-th}}$.

Analogously, PEM does not release candidate c if its count is below the threshold, i.e., $T[c]+\text{Laplace}(1/\epsilon) \le \tau_{\text{PEM}}+\text{Laplace}(1/\epsilon)$. Assuming data is distributed uniformly between groups, we have $T[c] = f_c/g$. Assuming expected noise and $z = f_{|C|\text{-th}}$, replacing f_c by $f_{k\text{-th}}$ as before, we arrive at $f_{k\text{-th}}/g \le f_{|C|\text{-th}} + \tau_{\text{PEM}}$, which is the right side of the inequality when multiplied with g. □

For fixed η, larger domain bit-length leads to larger group size g in $\mathcal{F}_{\text{PEM}}$. Since $\mathcal{F}_{\text{HH}}$ is independent of the domain size, it provides better accuracy in such cases, as the counts per value are not split among multiple groups. However, we want to keep t small and fixed for our MPC protocol, as $\mathcal{F}_{\text{HH}}$ requires t operations per datum in the worst case (decrement step). Fixed t reduces accuracy for increasing data sizes (Lemma 1); therefore, $\mathcal{F}_{\text{HH}}$ is better suited for small data sets (small n). Our empirical analysis in Section 5 confirms these observations.

3.4 Distributed Noise Generation

Sampling the noise for $\mathcal{F}_{\text{HH}}, \mathcal{F}_{\text{PEM}}$ with secure computation [51] is inefficient, as the parties have to securely evaluate expensive (floating or fixed point) operations [3][3]. It is more efficient to use distributed noise generation, by letting each party locally compute partial noises, which are securely combined, as often found in DP literature [2, 35, 47, 49]. Distributed noise generation is possible for distributions that are *infinitely divisible*, i.e., noise samples can be expressed as the sum of independent and identically distributed random variables. Both distributions used in our protocols, namely Laplace and Gumbel, are infinitely divisible [2, 21]. Thus, we can efficiently combine partial noise values: A random variable $X \sim \text{Laplace}(b)$ can be expressed as $\sum_{j=1}^{n}(Y_j^1 - Y_j^2)$ for $Y_j^1, Y_j^2 \sim \text{Gamma}(\frac{1}{n}, b)$, where the Gamma distribution with shape k, scale b has density $\text{Gamma}(x; k, b) = \frac{1}{\Gamma(k)b^k} x^{k-1} \exp(-\frac{x}{b})$ [2].

To avoid floating point numbers, which require secure computation overhead compared to integers [3], one can use the discrete Laplace distribution defined over integers. The discrete Laplace distribution is infinitely divisible and can be expressed as the difference of two Pólya random variables as noted by Goryczka et al. [47]. Recent works consider alternative Laplace noise representations on finite machines, e.g., [9, 10, 44], which we can leverage as well. The distributed noise representation does not affect our MPC efficiency as they are based on (integer) addition. Note that δ depends on a security parameter, associated with number representation in MPC, which we account for in Section 4.4. We discuss distributed noise generation for Gumbel noise in Appendix C.

[3]Given a uniform random number $r \in (0, 1]$ one can sample Laplace(b) as $\pm b \log(r)$ [51, Supplementary Material].

Table 1: Basic MPC protocols.

MPC protocol	*Output / Functionality*
$\text{EQ}(\langle a\rangle, \langle b\rangle)$	$\langle 1\rangle$ if $a = b$, else $\langle 0\rangle$
$\text{LE}(\langle a\rangle, \langle b\rangle)$	$\langle 1\rangle$ if $a \le b$, else $\langle 0\rangle$
$\text{ADD}(\langle a\rangle, \langle b\rangle)$	$\langle a + b\rangle$
$\text{AND}(\langle a\rangle, \langle b\rangle)$	$\langle a \cdot b\rangle$
$\text{NOT}(\langle a\rangle)$	$\langle 1 - a\rangle$
$\text{CondSwap}(\langle a\rangle, \langle b\rangle, \langle c\rangle)$	$\langle a\rangle$ if bit $c = 1$, else $\langle b\rangle$
$\text{Rec}(\langle a\rangle)$	Reconstruct secret a

4 MPC FOR DP HEAVY HITTERS

We describe details of our MPC protocols HH, PEM which realize the ideal functionalities $\mathcal{F}_{\text{HH}}, \mathcal{F}_{\text{PEM}}$ without a trusted party, and analyse their running time and security.

We use upper case letters to denote arrays in our protocol, and $A[j]$ denotes the j-th element in array A. We indicate Boolean values (in the form of a bit) with b_{state} (e.g., $b_{\text{match}} = 1$ indicates a match). The MPC subprotocols used in our protocol are listed in Table 1. While most of our computation can be represented with integers, our protocol uses fixed point numbers (scaled, truncated floats) to handle DP noise. Limited machine precision of floating point numbers can lead to privacy violations in the implementation of the Laplace mechanism [62], i.e., possible outcomes can differ between neighboring data sets due to irregularities in representing reals). These violations can be mitigated by careful truncation and rounding. We do not release noisy counts and do not use floating point numbers, nonetheless, similar attacks might exist without careful selection of fixed-point numbers.

4.1 HH: MPC of $\mathcal{F}_{\text{HH}}$

Instead of a map T, as in $\mathcal{F}_{\text{HH}}$, we use two arrays V, C, that store a value and its corresponding count at the same index. HH implements the different if-else branches of $\mathcal{F}_{\text{HH}}$ by using (secret) bits: b_{found} indicates if a value is already in V; $b_{\text{empty},j}$ indicates if we had no match ($\text{NOT}(b_{\text{found}})$) but index j is empty; and $b_{\text{decrement}}$ is true if we did not find a match and have no empty spots left. We employ the following optimizations to reduce the number of MPC protocols: Instead of using OR to combine bit b_{match} into b_{found} we add each bit b_{match} (which can be 1 at most once) to form b_{found} (which is 1 only if any match occurred) in line 7. This is beneficial, since ADD can be evaluated locally in secret sharing, i.e., without interaction, whereas arithmetic expression of OR is $a + b - a \cdot b$, and multiplications requires interaction between the parties (see also Appendix E). Similarly, we reduce the number of conditional swaps by directly using $b_{\text{decrement}}$ as a decrement value. Furthermore, we do not need to remove values associated with empty counts, saving additional swaps: We only use counts to check if a value is empty and if the value is matched (even with empty count), we set the new count to 1 (line 16), i.e., same as if we had not matched and found

Algorithm 1 Algorithm HH.

```
Input: User data $D$, distributed noises $\rho_p$ per party $p \in \mathcal{P}$, output size $k$, map size $t$, and DP threshold $\tau_{\mathsf{HH}}$.
Output: DP top-$k$.
 1: Initialize arrays $\langle V \rangle$, $\langle C \rangle$ of size $t$ with $\langle \perp \rangle$, $\langle 0 \rangle$, resp.
 2: for user datum $d \in D$ do  //Update counts $C$ for values $V$
 3:   Initialize $\langle b_{\text{found}} \rangle \leftarrow \langle 0 \rangle$ and $\langle i_{\text{empty}} \rangle \leftarrow \langle -1 \rangle$
 4:   for index $j \leftarrow 1$ to $t$ do
 5:     $\langle b_{\text{match}} \rangle \leftarrow \mathsf{EQ}(\langle d \rangle, \langle V[j] \rangle)$
 6:     $\langle b_{\text{empty}} \rangle \leftarrow \mathsf{LE}(\langle C[j] \rangle, \langle 0 \rangle)$
 7:     $\langle b_{\text{found}} \rangle \leftarrow \mathsf{ADD}(\langle b_{\text{found}} \rangle, \langle b_{\text{match}} \rangle)$
 8:     $\langle i_{\text{empty}} \rangle \leftarrow \mathsf{CondSwap}(\langle j \rangle, \langle i_{\text{empty}} \rangle, \langle b_{\text{empty}} \rangle)$
 9:     $\langle C[j] \rangle \leftarrow \mathsf{ADD}(\langle C[j] \rangle, \langle b_{\text{match}} \rangle)$
10:   end for
11:   $\langle b_{\neg\text{empty}} \rangle \leftarrow \mathsf{EQ}(\langle i_{\text{empty}} \rangle, \langle -1 \rangle)$
12:   $\langle b_{\text{decrement}} \rangle \leftarrow \mathsf{AND}(\langle b_{\neg\text{empty}} \rangle, \mathsf{NOT}(\langle b_{\text{found}} \rangle))$
13:   for index $j \leftarrow 1$ to $t$ do  //Conditional decrement
14:     $\langle b_{\text{empty},j} \rangle \leftarrow \mathsf{AND}(\mathsf{NOT}(\langle b_{\text{match}} \rangle), \mathsf{EQ}(\langle i_{\text{empty}} \rangle, \langle j \rangle))$
15:     $\langle c \rangle \leftarrow \mathsf{ADD}(\langle C[j] \rangle, \langle -b_{\text{decrement}} \rangle)$
16:     $\langle C[j] \rangle \leftarrow \mathsf{CondSwap}(\langle 1 \rangle, \langle c \rangle, \langle b_{\text{empty},j} \rangle)$
17:     $\langle V[j] \rangle \leftarrow \mathsf{CondSwap}(\langle d \rangle, \langle V[j] \rangle, \langle b_{\text{empty},j} \rangle)$
18:   end for
19: end for
20: for index $j \leftarrow 1$ to $t$ do  //DP thresholding on noisy $C$
21:   for party $p \in \mathcal{P}$ do
22:     $\langle C[j] \rangle \leftarrow \mathsf{ADD}(\langle C[j] \rangle, \langle \rho_p^j \rangle)$
23:   end for
24:   $\langle b_{\text{discard}} \rangle \leftarrow \mathsf{LE}(\langle C[j] \rangle, \langle \tau_{\mathsf{HH}} \rangle)$
25:   $\langle V[j] \rangle \leftarrow \mathsf{CondSwap}(\langle \perp \rangle, \langle V[j] \rangle, \langle b_{\text{discard}} \rangle)$
26: end for
27: Sort values $\langle V \rangle$ by their counts $\langle C \rangle$ descendingly  //Appendix F
28: return Rec($\langle V \rangle$)
```

an empty spot. HH sorts t candidates via secure merge sort (Appendix F). Note that HH sorts a small map, i.e., $t \ll n$, and the main effort is updating the map for n elements. We also implemented a version more suited for parallelization, denoted as $\mathsf{HH}_{\text{threads}}$ in our evaluation (Section 5): The loop steps in HH can be run in parallel, if we do not set i_{empty} in the first loop (as this requires locking). Thus, the main difference between HH and $\mathsf{HH}_{\text{threads}}$ is that we use an additional (non-parallelized) loop to set i_{empty}.

4.2 PEM: MPC of $\mathcal{F}_{\mathsf{PEM}}$

PEM implements $\mathcal{F}_{\mathsf{PEM}}$ by using array C to count candidate prefixes, where, e.g., $C[1]$ represents 0000 in the first step with $\gamma + \eta = 4$. The users themselves can track which indices correspond to candidate prefixes, simplifying the secure computation complexity. In the last round of PEM, less than $2^{\lceil \log k \rceil + \eta}$ iterations are required if $(b - \lceil \log k \rceil)/\eta$ is not an integer. We use this optimization in our implementation but omit it here for readability. Note that we sort the candidates and do not release noisy counts. Recall, unrestricted sensitivity ($\Delta > k$) is realized with Gumbel noise (see Section 3.2.3). Gumbel noise, unlike Laplace noise, is not DP by itself [32]; hence, we cannot release noisy counts which the parties could sort locally. Also, each party can remove its partial noise from the noisy count, requiring additional noise or secure noise sampling (see Section 4.5). If we are not interested in the order, i.e., which value is the i-th most frequent, the sorting step can be replaced by linear scan (to find the

Algorithm 2 Algorithm PEM.

```
Input: Noisy user reports $\widehat{\zeta_d^c}$ indicating if $d \in D$ has prefix $c$ (with distributed noise as in Section 3.4), output size $k$, domain bit-length $b$, prefix extension bit-length $\eta$, DP threshold $\tau_{\mathsf{PEM}}$, and distributed noises $\rho_p$ per party $p \in \mathcal{P}$ (for threshold).
Output: DP top-$k$.
 1: Split users in $g = \left\lceil \frac{b - \lceil \log k \rceil}{\eta} \right\rceil$ disjoint groups where $D = \bigcup_{i=1}^{g} D_i$
 2: for group $i \leftarrow 1$ to $g$ do
 3:   Initialize arrays $\langle S \rangle$, $\langle C \rangle$ of sizes $k, 2^{\lceil \log k \rceil + \eta}$ with zeros
 4:   Initialize array $\langle I \rangle \leftarrow \{\langle 1 \rangle, \ldots, \langle 2^{\lceil \log k \rceil + \eta} \rangle\}$
 5:   Initialize $\langle \rho_\tau \rangle \leftarrow \langle 0 \rangle$ and $\langle \tau \rangle \leftarrow \langle 0 \rangle$
 6:   for candidate $c \leftarrow 1$ to $2^{\lceil \log k \rceil + \eta}$ do
 7:     for user datum $d \in D_i$ do  //Gather prefix candidate counts
 8:       $\langle C[c] \rangle \leftarrow \mathsf{ADD}(\langle C[c] \rangle, \langle \widehat{\zeta_d^c} \rangle)$  //Prefix bit-length: $i \cdot \eta + \gamma$
 9:     end for
10:   end for
11:   Sort candidate indices $\langle I \rangle$ by their corresponding counts $\langle C \rangle$ descendingly  //Appendix F
12:   for party $p \in \mathcal{P}$ do
13:     $\langle \rho_\tau \rangle \leftarrow \mathsf{ADD}(\langle \rho_\tau \rangle, \langle \rho_p \rangle)$
14:   end for
15:   $\langle \tau \rangle \leftarrow \mathsf{ADD}(\mathsf{ADD}(\langle \tau_{\mathsf{PEM}} \rangle, \langle \rho_\tau \rangle), \langle C[2^{\lceil \log k \rceil + \eta}] \rangle)$
16:   for candidate $c \leftarrow 1$ to $k$ do  //DP thresholding on noisy $C$
17:     $\langle b_{\text{discard}} \rangle \leftarrow \mathsf{LE}(\langle C[c] \rangle, \langle \tau \rangle)$
18:     $\langle S[c] \rangle \leftarrow \mathsf{CondSwap}(\langle \perp \rangle, \langle I[c] \rangle, \langle b_{\text{discard}} \rangle)$
19:   end for
20:   return Rec($\langle S \rangle$)
21: end for
```

minimum count for the threshold), improving the complexity of this step from $O(c \log c)$ to c for $c = 2^{\lceil \log k \rceil + \eta}$ (leading to c instead of k iterations for thresholding in line 16 of Algorithm 2).

4.3 Running Time Complexity

We analyse the running time of our protocols HH, PEM w.r.t. the number of basic MPC protocols from Table 1. Addition is omitted, as the parties can compute it locally on secret shares (i.e., "for free") and we measure the running time of our implementation in Section 5. The complexity per protocol is listed in Appendix E, and is at most $O(l)$ for l-bit integers.

Theorem 4. HH *has complexity* $O(nt)$.

Proof. For each n values in D HH requires: First, t equality checks (EQ), comparisons (LE), and conditional swaps (CondSwap), to find matching values and look for an empty index. Then, one EQ, AND, and NOT operation to set bit $b_{\text{decrement}}$. For the DP threshold, t LE and CondSwap operations are used. Finally, HH sorts the small map, i.e., $O(t \log t)$, and reconstruct t counts. Note that n is the dominating factor as $t \ll n$, i.e., $nt > t \log t$. Overall, HH performs $O(nt)$ operations. □

Theorem 5. PEM *with sorting has complexity* $O(gc \log c)$*, and* PEM *without sorting has complexity* $O(gc)$*, where* $g = \left\lceil \frac{b - \lceil \log k \rceil}{\eta} \right\rceil$ *and* $c = 2^{\lceil \log k \rceil + \eta}$.

Proof. First, we consider PEM with sorting. For each group PEM sorts all c candidates which requires $O(c \log c)$ operations, and performs k comparisons (LE) and conditional swaps (CondSwap). Finally, k (sorted) indices are returned. Overall, PEM with sorting requires $O(c \log c)$ operations per group.

PEM, without sorting, requires c comparisons per group to find the lowest candidate count (used in the threshold). Then, PEM iterates over c elements per group (instead of k elements as with sorting). Finally, c indices and counts are returned and the parties can sort them themselves. Altogether, PEM without sorting requires $O(c)$ operations per group. □

Note that the summation of user reports per prefix candidates (line 8 in Algorithm 2) does not require any interaction between the computation parties, as addition can be computed locally.

4.4 Privacy

Beimel et al. [14, Lemma 2.12] stated the following composition theorem for secure computation of differentially private mechanisms:

Theorem 6. *Let Π be a protocol with one invocation of a black-box access to some function f. Let Π_f be a protocol that securely computes f with security parameter κ and coalitions of size up to t. Let Π' be as in Π, except that the call to f is replaced with the execution of Π_f. If Π is (ϵ, δ)-DP with coalitions of size up to t, and $\delta_\kappa = negl(\kappa)$ then Π' is $(\epsilon, (\exp(\epsilon) + 1)\delta_\kappa + \delta)$-DP with coalitions of size up to t.*

Thus, together with Theorems 1, 2, our secure implementations HH and PEM are $(\Delta\epsilon, (e^\epsilon + 1)\delta_\kappa + \delta)$- and $(\Delta\epsilon, (e^\epsilon + 1)\delta_\kappa + \frac{\delta}{4}(e^{\Delta\epsilon} + 1)(3 + \log(\Delta/\delta)))$-DP, resp. PEM requires multiple invocations but over disjoint subsets of the data.

4.5 Security

We consider the *semi-honest model* introduced by Goldreich [45] where corrupted protocol participants do not deviate from the protocol but gather everything created during the run of the protocol. Our protocols HH and PEM consists of multiple subroutines realized with MPC protocols listed in Table 1. To analyze a protocol's security, we apply the well-known *composition theorem* [45, Section 7.3.1]: MPC protocols using an ideal functionality remain secure if the ideal functionality is replaced with an MPC protocol implementing the functionality. We implement ideal functionalities $\mathcal{F}_{\text{HH}}, \mathcal{F}_{\text{PEM}}$ as HH, PEM with MPC frameworks MP-SPDZ [53] and SCALE-MAMBA [4] (see Section 5).

More formally, to prove semi-honest security we show the existence of a *simulator* Sim such that the distributions of the protocol transcript of *secure implementation* Π is computationally indistinguishable from simulated transcript using *ideal functionality* $\mathcal{F}$ produced in an "ideal world" with a trusted third party [45], [41, Def. 2.2]. Next, we formalize the ideal and real-world executions, ideal and real:

- $(\{\mathsf{VIEW}^i_\Pi\}_{i \in C}, \{y_i\}_{i \in \mathcal{P}}) \leftarrow \text{real}_\Pi(\kappa, C, \{x_i\}_{i \in \mathcal{P}})$, receives as input security parameter κ, the set $C \subset \mathcal{P}$ of corrupted parties, and each parties input x_i. Then, the real-world execution runs protocol Π, with each party $i \in \mathcal{P}$ behaving honestly using its own input x_i, and outputs the view of all corrupted parties (i.e., all exchanged messages and internal state), as well as the final output y_i of each party.
- $(\mathcal{S}, \{y_i\}_{i \in \mathcal{P}}) \leftarrow \text{ideal}_{\mathcal{F},\text{Sim}}(\kappa, C, \{x_i\}_{i \in \mathcal{P}})$, with the same inputs, uses the ideal functionality $\mathcal{F}$ to compute $\{y_i\}_{i \in \mathcal{P}} \leftarrow \mathcal{F}(\{x_i\}_{i \in \mathcal{P}})$. Then, the ideal-world execution runs simulator $\mathcal{S} \leftarrow \text{Sim}(C, \{(x_i, y_i)\}_{i \in C})$ (i.e., simulator receives the set of corrupted parties and their in/outputs) to create simulation $\mathcal{S}$, and output it along with the output of the parties.

An adversary in the ideal world learns nothing except the protocol inputs and outputs, hence, if he cannot distinguish simulated transcripts (ideal world) from actual transcripts (real world), he learns nothing in real-world implementations. Now we show the existence of simulators for our protocols.

Theorem 7. *Protocol* HH *realizes $\mathcal{F}_{\text{HH}}$ in the presence of semi-honest adversaries.*

Proof. Simulator Sim, given final outputs V, C (i.e., $\{y_i\}_{i \in \mathcal{P}}$) can produce a transcript for real_{HH} by replacing all secret shared values with randomness. Note that all values in our protocols are secret shared (marked with $\langle \cdot \rangle$) and computationally indistinguishable from randomness (except with negligible probability in the security parameter for some operations, e.g., integer comparisons [4]). The only values that are not secret shared are publicly known iteration counts (i.e., data size and map size t for HH, and number of groups and number of candidates in PEM). Finally, the simulator ensures the expected reconstruction, i.e, V, C, is produced by $\text{Rec}(V), \text{Rec}(C)$. Here, the corrupted parties, cannot distinguish actual from simulated reconstruction as they cannot see the actual randomness (secret shares) from the other parties. □

Theorem 8. *Protocol* PEM *realizes $\mathcal{F}_{\text{PEM}}$ in the presence of semi-honest adversaries.*

Proof. We focus on a transcript for one group of PEM, which can be extended to all groups. Simulator Sim, given S, produces a transcript of real_{PEM} as follows: As before, Sim replaces all secret shared values with randomness. Then, in the thresholding step, the index for each candidate c, i.e., $S[c]$ is set such that the reconstruction of S provides the expected result. □

From Semi-honest to Malicious: We consider semi-honest computation parties and design our protocol accordingly. However, SCALE-MAMBA provides malicious security, i.e., consistency within the computation is ensured and malicious tampering can be detected. We employ (t, m)-secret sharing, which prevents up to $t - 1$ malicious parties to reconstruct the secret. Still, malicious parties (input parties or computation servers) can provide incorrect initial inputs to skew the results, also known as a *data poisoning attack*. Next, we discuss the affect of poisoning attacks on our protocol as well as potential (but not implemented) mitigations. In general, LDP protocols are vulnerable to data poisoning attacks [22, 24]. Cryptographic tools, however, can prevent data poisoning attacks and such attacks have limited impact on our protocols HH and PEM: For HH, each input party provides a single value, which can change a count by at most 1; thus, a coalition of c malicious parties, can alter the count by at most c. For PEM, each input party provides a single bit indicating if a prefix matches their value's prefix (1) or not (0). Thus, c malicious parties, can skew the result by at most c. (However, this requires additional zero-knowledge proofs, ensuring that the provided value is from $\{0, 1\}$ without revealing it, e.g., [28, 65].) Distributed noise generation in the presence of malicious parties is not possible without additional checks or assumptions. E.g., assuming c malicious parties, all parties have to provide more noise[4] as the malicious parties might not provide any noise (or hide additional counts in the noise). To achieve optimal noise magnitudes in the presence of malicious parties the Laplace noise can be sampled securely: Given a uniform random $r \in (0, 1]$[5] one can sample Laplace(b) as $\pm b \log(r)$. However, this incurs additional computation costs [5], which we do not consider, since we assume semi-honest parties like most LDP protocols [11, 12, 40, 42].

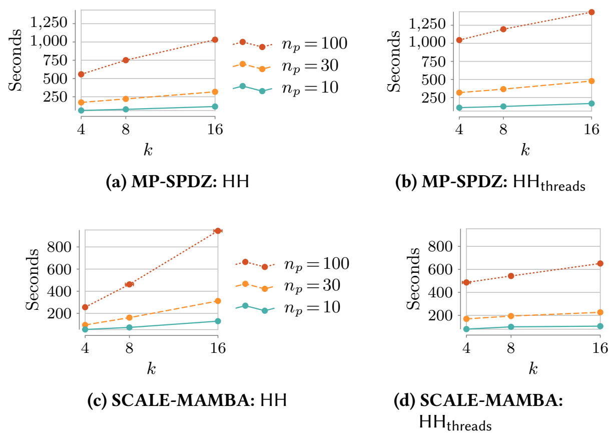
Outsourcing: To outsource the computation the n *input parties* send shares of their input to m *computation parties* which run the secure computation on their behalf. The latter can be a subset of the input parties or non-colluding untrusted servers (e.g., multiple cloud service providers). After sending their secret shared value for HH or candidate counts for PEM the input parties can go offline.

5 EVALUATION

We implement our protocols with SCALE-MAMBA [4] (malicious security) as well as MP-SPDZ [53] (semi-honest security) using Shamir secret sharing with honest majority, and default settings, i.e., 128-bit modulus and statistical security parameter $\kappa = 40$. Code can be largely re-used between these frameworks as MP-SPDZ [53] is a fork of SCALE-MAMBA's predecessor SPDZ2.

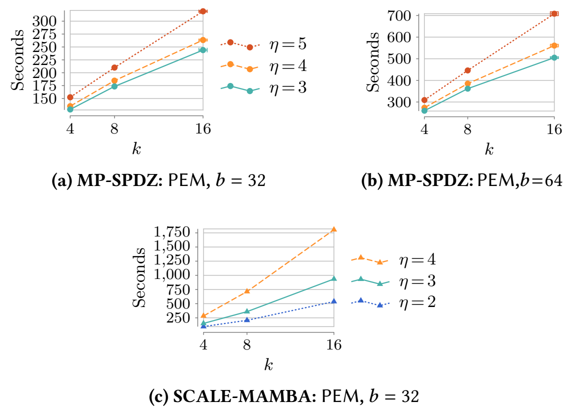
We evaluated the running time and communication of the *entire* protocol, i.e., offline as well as online phase, in a real-world WAN for $m = 3$ parties. We split the parties into two AWS regions, Ohio (us-east-2) and Frankfurt (eu-central-1), and measured an inter-region round time trip (RTT) of approx. 100 ms with 100 Mbits/s bandwidth. The computation parties already received and combined secret-shared inputs from the input users (Section 4.5). We present the average of 10 runs for running time and communication (except MP-SPDZ for $\text{HH}_{\text{threads}}$ with 3 runs) and 20 runs for accuracy with 95% confidence intervals, but omit the intervals in most cases, as the results are very stable. We used modest hardware, t2.medium AWS instances (2 GB RAM, 4vCPUs) [6], to show that the computational overhead of modern MPC is acceptable. (More powerful hardware did not provide significant improvements.) Recall, $\text{HH}_{\text{threads}}$ is a parallelized version of HH (Section 4.1), which required c4.2xlarge instances (15 GB RAM, 8vCPUs) to leverage 8 threads. Also, t2.large (8 GB RAM, 4vCPUs) instances were used for PEM in two settings – MP-SPDZ with $\eta = 5, k = 16$, and SCALE-MAMBA with $\eta = 4, k = 16$ – as more memory was required for these larger programs. To evaluate running time and communication of HH, we set map size $t = k$, and fix it to 16 in our accuracy evaluation (Section 5). We stress that we evaluated a worst-case scenario for PEM: Each round assumes that the maximum of k prefix candidates are output after thresholding. Fewer outputs decrease computation and communication due to smaller candidate sets for the next round. We securely sort the candidates. However, if one is not interested in the order, i.e., which value is the i-th most frequent, the sorting step can be replaced by linear scan (to find the minimum count for the threshold). Next, we evaluate accuracy, running time and communication of our protocols in a real-world WAN. Evaluation of AWS cost is given in Appendix G, further comparisons of SCALE-MAMBA with MP-SPDZ are detailed in Appendix K.

Running Time: Figures 5, 6 show the running times for HH, PEM implemented with MP-SPDZ as well as SCALE-MAMBA with data sizes $n_p \in \{10, 30, 100\}$ *per computation party* p (i.e., $|D| \in \{30, 90, 300\}$). To show the difference between HH and $\text{HH}_{\text{threads}}$, we used the same scale for MP-SPDZ (Figures 5a, 5b) and SCALE-MAMBA (Figures 5c, 5d). For MP-SPDZ, the running time with 8 threads increases, whereas it decreases with SCALE-MAMBA. Overall, for HH, and especially $\text{HH}_{\text{threads}}$, SCALE-MAMBA is faster than MP-SPDZ, requiring at most 11 minutes for $\text{HH}_{\text{threads}}$, and less than 16 for HH. The opposite is the case for PEM: MP-SPDZ is much faster, taking less than 6 minutes for $\eta = 5$, whereas SCALE-MAMBA requires almost half an hour for $\eta = 4$. Note that we used smaller values of η for SCALE-MAMBA (i.e., $\eta \in \{2, 3, 4\}$) since the differences to MP-SPDZ are already sufficiently pronounced here.



(a) **MP-SPDZ:** HH (b) **MP-SPDZ:** $\text{HH}_{\text{threads}}$ (c) **SCALE-MAMBA:** HH (d) **SCALE-MAMBA:** $\text{HH}_{\text{threads}}$

Figure 5: Running time of HH, $\text{HH}_{\text{threads}}$**.**



(a) **MP-SPDZ:** PEM, $b = 32$ (b) **MP-SPDZ:** PEM,b=64 (c) **SCALE-MAMBA:** PEM, $b = 32$

Figure 6: Running time of PEM**.**

[4]See Ács et al. [2, Section 8.3] (technical report version) for a detailed analysis of the required noise increase.

[5]Uniform random numbers can be generated in a distributed manner even in a malicious setting, e.g., by XORing random inputs from each party (which is random as long as a single party provides actual randomness) [51, Supplementary Material], or by using the randomness generated in the offline phase [4].

Client Communication: For HH, a client (input party) sends her secret-shared value to each of the m servers (computation parties). In total, a client sends $m \cdot 128$ bits (our evaluated share size is 128 bits). For PEM, a client sends $2^{\lceil \log k \rceil + \eta}$ secret-shared counts, i.e., at most $m \cdot 8\,KB$ (our largest evaluation with $\eta = 5, k = 16$).

Server Communication: As is to be expected, semi-honest MP-SPDZ sends less than maliciously secure SCALE-MAMBA. We briefly evaluated MP-SPDZ with malicious security (Appendix K), and found it to be still more communication-efficient, albeit slower, than SCALE-MAMBA. Next, we report the average communication of HH and PEM per party for $k = 16$. Further evaluations are provided in Appendix H. For HH communication increases linearly with data size. We consider data size n_p per computation party $p \in \{1, 2, 3\}$, and MP-SPDZ requires ≈13/38/122 MB for n_p 10/30/100. While SCALE-MAMBA provides better running times than MP-SPDZ for HH$_{\text{threads}}$, MP-SPDZ requires much less communication, e.g., roughly 45 times less for HH$_{\text{threads}}$ with $k = 16, n_p = 100$ (125 MB vs 5.6 GB), suggesting superior communication batching and parallelization from SCALE-MAMBA compared to MP-SPDZ. For PEM and $b = 32$, MP-SPDZ sends ≈130/258 MB and SCALE-MAMBA sends ≈989/1884 MB for η 3/4. Doubling the domain bit-length to 64 also roughly doubles the communication. Note that PEM, unlike HH, is independent of the data size, as we now consider aggregated candidate counts and not single values.

Comparing different DP notions: We use the same value for ϵ to compare our approach to state-of-the-art PEMorig for heavy hitter detection in the local model. Our protocols, however, operate in the central model realized with MPC and approximate differential privacy ($\delta > 0$), whereas PEMorig is a local model protocol with pure differential privacy ($\delta = 0$). The main benefit of approximate DP is improved composition [37, Section 3.5], i.e., running g mechanisms on the same data requires a smaller privacy budget of $\approx \sqrt{g}\epsilon$ instead of $g\epsilon$ for large enough g. However, we run PEM once per disjoint subsets of the data and not multiple times on the same data. Thus, we gain no significant advantage over PEMorig from using approximate DP. Furthermore, for an advantage to become noticeable one requires $g > \frac{2\log(1/\delta)}{(2-\exp(\epsilon)^2}$ (Appendix D, [64]).

Accuracy: Next, we evaluate accuracy via NCR score (Definition 2), and provide a comparision to F1 in Appendix J. For the accuracy evaluation, we set $\Delta = 1, \delta = 10^{-7}$, assume domain bit-length $b = 32$, and report the average of 20 runs with 95% confidence intervals. Like Wang et al. [78], we use a synthetic data set sampled from the Zipf distribution with parameter 1.5, i.e., the j-th most frequent value appears with probability proportional to $1/j^{1.5}$. We also used prices from an Online retail data set [75]. Note that we use small data sizes (few thousand values) on purpose, as it is the most challenging regime for DP, i.e., the ratio of "signal" (actual counts) to noise is small. We compare PEM and PEMorig for different values of $\eta \in \{4, 5\}$, where η is given in brackets (e.g., "PEM($\eta = 4$)"), as well as with PEMorig with query limit count of 2^{20} (denoted as "PEMorig"), where η is set to the largest integer satisfying $g2^{\gamma+\eta} < 2^{20}$ for $g = \lceil (b - \gamma)/\eta \rceil$ groups and $\gamma = \lceil \log_2 k \rceil$ as suggested [78].

In Figure 8 we fix $\epsilon = 2, n = 1000$ and vary $k \in \{4, 8, 16\}$. As expected, when we increase k while keeping n fixed (and small), the accuracy decreases for all evaluated approaches. However,

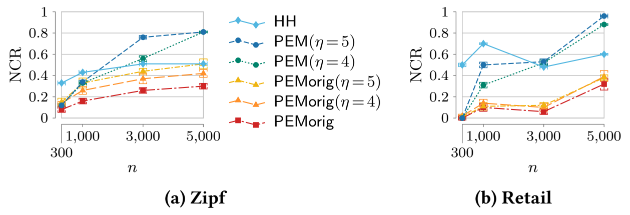


Figure 7: NCR of PEM **variants and** HH **for fixed** $\epsilon = 2, k = 16,$ **varying** $n \in \{300, 1000, 3000, 5000\}$**.**

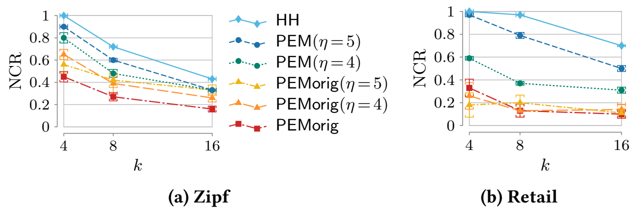


Figure 8: NCR of PEM **variants and** HH **for fixed** $\epsilon = 2, n = 1000$**, varying** $k \in \{4, 8, 16\}$**.**

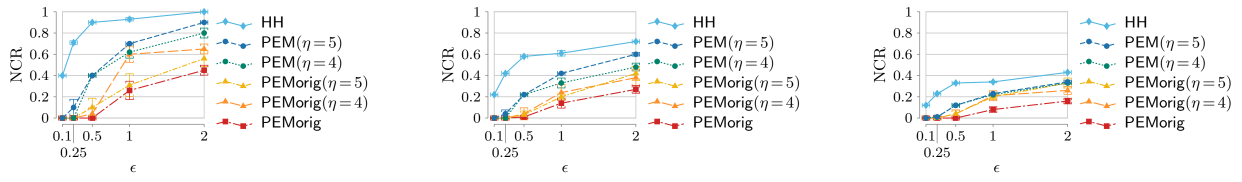
as shown in Figure 7 – where we fix $k = 16, \epsilon = 2$ and vary $n \in \{300, 1000, 3000, 5000\}$ – increasing the data size improves accuracy, as the candidates receive more counts, which can more easily surpass the DP thresholds.

Figures 9, 11, show NCR for PEM with data size 1000 for $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$, where we vary $k \in \{4, 8, 16\}$. Figures 10, 12 show the same for data size 5000. First, we focus on comparing PEM with PEMorig. Figure 9c shows that for large k (16) and small Zipf data size (1000), the difference between all approaches is not too strong, still HH, PEM provide better results. However, when we increase the data size (5000) in Figure 10c the accuracy of PEM rises much faster than PEMorig (and its variations with fixed η). We make the same observation, with the real-world data set in Figures 11c, 12c, i.e., PEM is more accurate and its accuracy improves faster when the data size increases. Overall, PEM provides higher accuracy than PEMorig.

Next, we fix $t = 16$ and compare HH to PEM. The choice of map size t provides the following trade-off: keeping t fixed (to a small value) while increasing k decreases accuracy; however, small values for t provide better efficiency for our MPC protocol. With data size 1000 (Figures 9, 11) HH provides the best accuracy. For data size 5000 (Figures 10, 12) and $k = 4$, HH still provides the best accuracy, however, PEM improves upon HH for $k > 4$. Altogether, the empirical evaluation confirms our analysis in Section 3.3: HH provides better accuracy for small data sizes with modest values for t. Also, PEM provides better accuracy than PEMorig for larger data sizes $n \in 10^5, 2 \cdot 10^5, 5 \cdot 10^5$ as we detail in Appendix I.

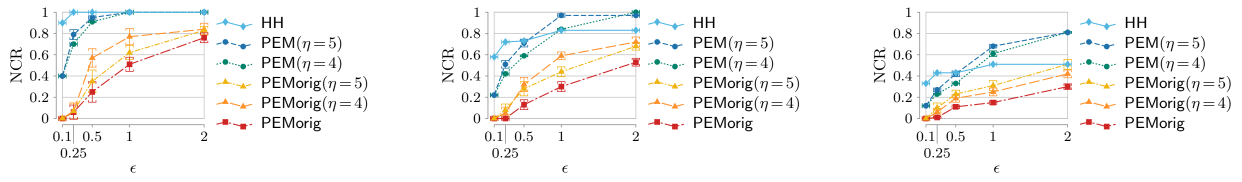
6 RELATED WORK

Non-private top-k: Algorithms for heavy hitter detection are roughly grouped into three classes [7, 27]: *Quantile algorithms*, which uses estimated quantiles of range endpoints to approximate



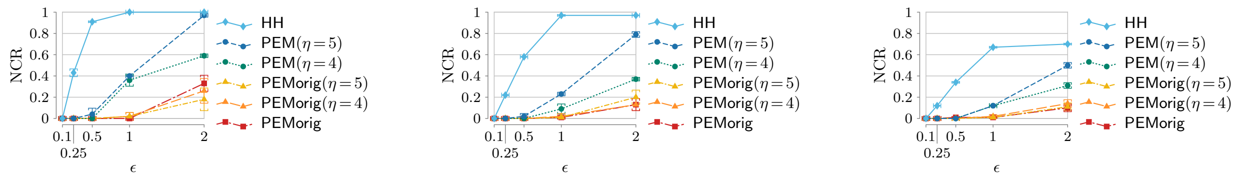
(a) $k = 4$ **(b)** $k = 8$ **(c)** $k = 16$

Figure 9: NCR of PEM **variants and** HH **for Zipf with fixed** $n = 1000$**, and varying** $k \in \{4, 8, 16\}$**,** $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$**.**



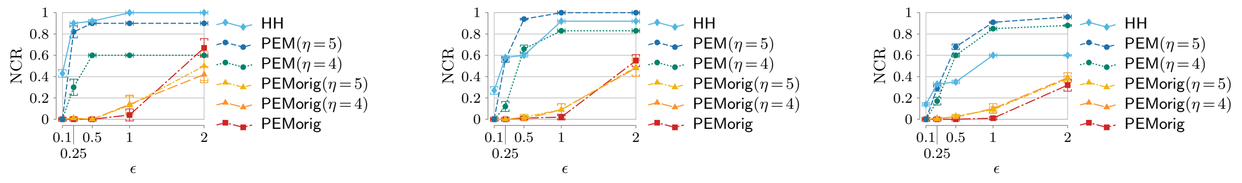
(a) $k = 4$ **(b)** $k = 8$ **(c)** $k = 16$

Figure 10: NCR of PEM **variants and** HH **for Zipf with fixed** $n = 5000$**, and varying** $k \in \{4, 8, 16\}$**,** $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$**.**



(a) $k = 4$ **(b)** $k = 8$ **(c)** $k = 16$

Figure 11: NCR of PEM **variants and** HH **for retail data [75] with fixed** $n = 1000$**, varying** $k \in \{4, 8, 16\}$**,** $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$**.**



(a) $k = 4$ **(b)** $k = 8$ **(c)** $k = 16$

Figure 12: NCR of PEM **variants and** HH **for retail data [75] with fixed** $n = 5000$**, varying** $k \in \{4, 8, 16\}$**,** $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$**.**

frequencies of range elements; *hash-based sketches*, which provide a space-efficient frequency estimation, and *counter-based sketches*, where a set of counters are updated when new data arrives. The latter are the best with regards to space, speed and accuracy [7, 27], thus, we selected [27, Alg. 1] as basis for HH. HH provides differentially privacy unlike related work [7, 27]. While recent improvements achieve better performance [7] (amortized over the control flow), we cannot leverage them in HH due to our use of MPC (which hides the control flow).

Local DP top-k: LDP heavy hitter approaches [11, 12, 40, 42, 78, 81] mainly differ in how they encode and reconstruct the candidates, for which counts are estimated. Such encoding (in the form of domain reduction, e.g., Bloom filters [40], matrix projection [12]) incurs information loss, which can exceed the loss due to DP randomization [78]. Notably, some encodings already provide some form of DP, e.g., [81] (or [26] for distinct counts), but only with large ϵ or for large data sizes. Wang et al. [78] carefully analyze related work [11, 12, 40, 42], which mainly utilize non-overlapping segments (e.g., report single bits or sets of bits), present a state-of-the-art protocol by leveraging overlapping prefixes, and show that it provides better accuracy than other approaches. We use [78] as basis for our central model protocol PEM, which does not suffer from information loss due to encoding and local randomization, and allows (central model) $O(1)$ count accuracy instead of (local model) $\Omega(\sqrt{n})$ for n users [23]. Also, we directly output heavy hitters (as our sketches contain their values or bit representation); unlike related work, where costly reconstructions are required to find heavy hitters given an encoded representation (e.g, hash), which has to be mapped to potential candidates from the domain [40, 42, 78].

Central DP top-k: An alternative to approximate DP with thresholding is probabilistic selection with pure DP, i.e., via exponential mechanism [60] or report noisy max [37]. These alternatives can be applied in a *peeling* fashion to find the most frequent value from a known domain, remove it from the domain, and repeat until k values are found. More computationally efficient *one-shot* methods [38, 70] release k values in one go. We choose thresholding as it is preferable, especially for small data, for two reasons: First, selection requires considering *all elements from a known domain* and sampling an output from the entire domain with probability proportional to an element's utility. With thresholding, on the other hand, focusing on *data elements (from an unknown domain)* suffices – leveraged by our protocol HH. Second, for large domains (e.g., of size 2^{32}) and small data (e.g., few hundred elements) the probability mass of elements with count zero (i.e., not in the data but in the domain) can exceed the selection probability of even the most frequent element, which destroys accuracy (especially using disjoint groups that split the counts among them). Durfee and Rogers [33] first compute the actual top-k', where $k' > k$, and use $(\epsilon, 0)$-DP noise and δ-based thresholding to release (ϵ, δ)-DP top-k. All central DP approaches assume access to the raw data or a trusted third party. We, on the other hand, securely discover top-k' without such assumptions, and apply thresholding [33] to release DP top-k in PEM. Vadhan [76, Theorem 3.5] presents a stability-based algorithm for central DP histograms and HH can be seen as a space-efficient version in the distributed setting.

MPC DP top-k: Melis et al. [61] combine count-min and count sketches as follows: parties evaluate multiple hash functions on their input, set the counters indexed by the hash functions to 1, and securely aggregate the counters. They mention heavy hitters as an application but do not evaluate. However, reconstructing heavy hitters from such sketches is linear in the domain size (as each candidate is mapped to sketch entries by evaluating multiple hash functions), whereas our protocols are linear in the data size (HH) or linear in the domain bit-length (PEM) and efficiently handle large and even unknown domains. Additional data structures, e.g., using multiple sketches [20], reduce the reconstruction complexity at the cost of increasing communication and aggregation overhead.

Boneh et al. [20] securely compute heavy hitters in a malicious setting with two computation servers. They focus on novel cryptographic primitives, i.e., incremental distributed point functions, allowing secret shares of size $O(m)$ to represent a vector of 2^m values with only one non-zero element. They consider DP only optionally to bound their protocol's information leakage. In contrast, DP with high accuracy is at the heart of our design, whereas they require large noise addition from each server, prohibiting any meaningful DP statistics for small number of clients and overall provide less accuracy than our DP-focused protocols. They require millions of clients to achieve an absolute error of 16% for $\epsilon < 1$ [20, Appendix E] and add noise multiple times and not per group. While their server communication is more efficient than ours (requiring only kilobytes), we have similar client communication (kilobytes), however, their computation time is linear in the number of parties. PEM is linear in the domain bit-length and asymptotically faster than Boneh et al. [20]. Adjusted for $k = 256, b = 256$, PEM is faster than their approach for more than 6 million clients[6].

Naor et al. [65] consider DP collection of frequently used passwords with malicious parties. On a very high-level, their hash-then-match approach is similar to PEMorig with $n2^l$ server operations, albeit more efficient ones (no hashing): Each user j receives a random l-bit value r_j from the server, computes l-bit hash h_j of her password and reports one bit, the inner product of r_j and $\mathrm{GRR}(h_j)$ over $GF[2]$. The server keeps 2^l counters, tries to find a matching $x \in 2^l$ for every report and increments the corresponding counters. Hash values are released if their noisy counts exceed a fixed fraction of the user count. Almost an LDP protocol, with the same accuracy limitations, where secure computation is required as malicious users cannot learn r_j. Their protocol is a series of two-party computations between users and server, whereas our protocol is a multi-party computation, where users can outsource the computation and only need to secret share their inputs.

7 CONCLUSION

We presented protocols for federated, differentially private top-k discovery with secure multi-party computation. Our central DP approaches, HH and PEM, provide higher accuracy than local DP methods for small number of users, without a trusted third party due to our use of cryptography. HH, based on non-private heavy hitter discovery in data streams [27], has a running time liner in the data size but supports unknown domains, and provides better accuracy than PEM for very small data sizes, where local DP methods cannot provide meaningful accuracy. PEM, based on Wang et al. [78], iteratively finds and extends frequent prefixes, is linear in the bit-length of the data domain, and provides better accuracy than HH for larger data sizes. We implemented our protocols with two MPC frameworks [4, 53], compared them, and achieved practical running times of less than 11 minutes in a real-world WAN. Future work for MPC sketching might reduce reconstruction overhead by combining multiple sketches (at the cost of increased communication). Also, one might leverage public data like Li et al. [58] and consider adaptive composition [31] instead of parallel composition.

Acknowledgments

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

[6] Boneh et al. [20, Table 9] process ≈120 clients/second (on 32 vCPUs, 60 GB RAM, ≈62 ms WAN delay, $b = 256$ and k as 0.1% of number of clients). PEM runs in less than 12 minutes in total (on 4 vCPUs, 8 GB RAM with 100 ms delay, $b = 64, k = 16$) and is independent of client count n, but linear in k and b. PEM's time multiplied by 256/16 (adjusts k), 256/64 (b), and 120 clients/s, results in ≈ 5.6 million clients.

REFERENCES

[1] John M. Abowd. 2018. The U.S. Census Bureau Adopts Differential Privacy. In *Proceedings of the annual ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD).*

[2] Gergely Ács and Claude Castelluccia. 2011. I have a dream! (differentially private smart metering). In *International Workshop on Information Hiding (IH).*

[3] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. 2013. Secure Computation on Floating Point Numbers. In *Network and Distributed Systems Security Symposium (NDSS).*

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

[5] Abdelrahaman Aly and Nigel P Smart. 2019. Benchmarking Privacy Preserving Scientific Operations. In *International Conference on Applied Cryptography and Network Security (ACNS).*

[6] Amazon.com. 2020. Amazon Web Services. https://aws.amazon.com/ec2/pricing/on-demand/.

[7] Daniel Anderson, Pryce Bevan, Kevin Lang, Edo Liberty, Lee Rhodes, and Justin Thaler. 2017. A high-performance algorithm for identifying frequent items in data streams. In *Proceedings of the Internet Measurement Conference (ICM).*

[8] Apple. 2016. WWDC 2016: Engineering Privacy for Your Users. Retrieved October, 2020 from https://developer.apple.com/videos/play/wwdc2016/709/ https://developer.apple.com/videos/play/wwdc2016/709/.

[9] Victor Balcer and Salil Vadhan. 2017. Differential privacy on finite computers. *arXiv preprint arXiv:1709.05396* (2017).

[10] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. 2020. Private summation in the multi-message shuffle model. In *Proceedings of the annual ACM conference on computer and communications security (CCS).*

[11] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. 2017. Practical locally private heavy hitters. In *Advances in Neural Information Processing Systems (NeurIPS).*

[12] Raef Bassily and Adam Smith. 2015. Local, private, efficient protocols for succinct histograms. In *Proceedings of the annual ACM symposium on Theory of computing (STOC).*

[13] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference (CRYPTO).*

[14] Amos Beimel, Iftach Haitner, Kobbi Nissim, and Uri Stemmer. 2020. On the Round Complexity of the Shuffle Model. *arXiv preprint arXiv:2009.13510* (2020). https://arxiv.org/abs/2009.13510 https://arxiv.org/abs/2009.13510.

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

[16] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the Symposium on Operating Systems Principles (SOSP).*

[17] Google Developers Blog. 2019. Enabling developers and organizations to use differential privacy. Retrieved October, 2020 from https://developers.googleblog.com/2019/09/enabling-developers-and-organizations.html https://developers.googleblog.com/2019/09/enabling-developers-and-organizations.html.

[18] Jonas Böhler and Florian Kerschbaum. 2020. Secure Multi-party Computation of Differentially Private Median. In *USENIX Security Symposium (USENIXSec).*

[19] Jonas Böhler and Florian Kerschbaum. 2020. Secure Sublinear Time Differentially Private Median Computation. In *Network and Distributed Systems Security Symposium (NDSS).*

[20] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. (2021).

[21] Pierre Bosch and Thomas Simon. 2013. On the self-decomposability of the Fréchet distribution. *Indagationes Mathematicae* (2013).

[22] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2019. Data Poisoning Attacks to Local Differential Privacy Protocols. *arXiv preprint arXiv:1911.02046* (2019). https://arxiv.org/abs/1911.02046.

[23] TH Hubert Chan, Elaine Shi, and Dawn Song. 2012. Optimal lower bound for differentially private multi-party aggregation. In *European Symposium on Algorithms (ESA).*

[24] Albert Cheu, Adam Smith, and Jonathan Ullman. 2019. Manipulation attacks in local differential privacy. *arXiv preprint arXiv:1909.09630* (2019). https://arxiv.org/abs/1909.09630.

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

[26] Seung Geol Choi, Dana Dachman-Soled, Mukul Kulkarni, and Arkady Yerukhimovich. 2020. Differentially-Private Multi-Party Sketching for Large-Scale Statistics. In *Proceedings on Privacy Enhancing Technologies (PETS).*

[27] Graham Cormode and Marios Hadjieleftheriou. 2010. Methods for finding frequent items in data streams. *The VLDB Journal* (2010).

[28] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, robust, and scalable computation of aggregate statistics. *USENIX Symposium on Networked Systems Design and Implementation.*

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

[30] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Advances in Neural Information Processing Systems (NeurIPS).*

[31] Jinshuo Dong, David Durfee, and Ryan Rogers. 2020. Optimal Differential Privacy Composition for Exponential Mechanisms. In *International Conference on Machine Learning (ICML).*

[32] David Durfee and Ryan Rogers. 2019. Practical Differentially Private Top-k Selection with Pay-what-you-get Composition. *arXiv preprint arXiv:1905.04273* (2019). (Extended version). https://arxiv.org/abs/1905.04273.

[33] David Durfee and Ryan Rogers. 2019. Practical Differentially Private Top-k Selection with Pay-what-you-get Composition. In *Advances in Neural Information Processing Systems (NeurIPS).*

[34] Cynthia Dwork. 2006. Differential Privacy. In *International Colloquium on Automata, Languages, and Programming (ICALP).*

[35] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT).*

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

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

[38] Cynthia Dwork, Weijie Su, and Li Zhang. 2015. Private false discovery rate control. *arXiv preprint arXiv:1511.03803* (2015). https://arxiv.org/abs/1511.03803.

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

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

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

[42] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. 2016. Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries. *Proceedings on Privacy Enhancing Technologies* (2016).

[43] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. 2016. Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries. *arXiv preprint arXiv:1503.01214* (2016). (Extended Version). https://arxiv.org/pdf/1503.01214.pdf.

[44] Badih Ghazi, Ravi Kumar, Pasin Manurangsi, and Rasmus Pagh. 2020. Private counting from anonymous messages: Near-optimal accuracy with vanishing communication overhead. In *International Conference on Machine Learning (ICML).*

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

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

[47] Slawomir Goryczka and Li Xiong. 2017. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE Transactions on Dependable and Secure Computing* (2017).

[48] Daniel H Greene and Donald E Knuth. 2007. *Mathematics for the Analysis of Algorithms.* Springer Science & Business Media.

[49] Mikko Heikkilä, Eemil Lagerspetz, Samuel Kaski, Kana Shimizu, Sasu Tarkoma, and Antti Honkela. 2017. Differentially private Bayesian learning on distributed data. In *Advances in Neural Information Processing Systems (NeurIPS).*

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

[51] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. 2018. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *Advances in Neural Information Processing Systems (NeurIPS).*

[52] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* (2011).

[53] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the annual ACM conference on Computer and Communications Security (CCS).*

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

[55] Marcel Keller, Dragos Rotaru, Nigel P Smart, and Tim Wood. 2018. Reducing communication channels in MPC. In *International Conference on Security and*

Cryptography for Networks (SCN).

[56] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. 2016. Differential Privacy: From Theory to Practice. *Synthesis Lectures on Information Security, Privacy, & Trust* (2016).

[57] Yehuda Lindell and Benny Pinkas. 2009. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology* (2009).

[58] Terrance Liu, Giuseppe Vietri, Thomas Steinke, Jonathan Ullman, and Zhiwei Steven Wu. 2021. Leveraging Public Data for Practical Private Query Release. *arXiv preprint arXiv:2102.08598* (2021). https://arxiv.org/pdf/2102.08598.pdf.

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

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

[61] Luca Melis, George Danezis, and Emiliano De Cristofaro. 2016. Efficient Private Statistics with Succinct Sketches. (2016).

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

[63] Jayadev Misra and David Gries. 1982. Finding repeated elements. *Science of computer programming* (1982).

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

[65] Moni Naor, Benny Pinkas, and Eyal Ronen. 2019. How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior. In *Proceedings of the annual ACM conference on Computer and Communications Security (CCS).*

[66] Seth Neel, Aaron Roth, Giuseppe Vietri, and Zhiwei Steven Wu. 2020. Oracle Efficient Private Non-Convex Optimization. (2020). https://proceedings.icml.cc/static/paper_files/icml/2020/354-Paper.pdf.

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

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

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

[70] Ryan Rogers. 2020. A Differentially Private Data Analytics API at Scale. In *USENIX Conference on Privacy Engineering Practice and Respect (PEPR).*

[71] Ryan Rogers, Subbu Subramaniam, Sean Peng, David Durfee, Seunghyun Lee, Santosh Kumar Kancha, Shraddha Sahay, and Parvez Ahammad. 2020. LinkedIn's Audience Engagements API: A Privacy Preserving Data Analytics System at Scale. *arXiv preprint arXiv:2002.05839* (2020). https://arxiv.org/abs/2002.05839.

[72] Adi Shamir. 1979. How to share a secret. *Commun. ACM* (1979).

[73] Hassan Takabi, Samir Koppikar, and Saman Taghavi Zargar. 2016. Differentially Private Distributed Data Analysis. In *IEEE International Conference on Collaboration and Internet Computing (CIC).*

[74] Apple's Differential Privacy Team. 2017. Learning with Privacy at scale. https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html.

[75] Machine Learning Group ULB. 2019. *Online Retail.* Retrieved October, 2020 from https://archive.ics.uci.edu/ml/datasets/Online+Retail+II https://archive.ics.uci.edu/ml/datasets/Online+Retail+II.

[76] Salil Vadhan. 2017. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography.* Springer.

[77] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally differentially private protocols for frequency estimation. In *USENIX Security Symposium (USENIXSec).*

[78] Tianhao Wang, Ninghui Li, and Somesh Jha. 2019. Locally differentially private heavy hitter identification. *IEEE Transactions on Dependable and Secure Computing* (2019).

[79] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. 2020. Differentially private SQL with bounded user contribution. In *International Symposium on Privacy Enhancing Technologies Symposium (PETS).*

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

[81] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. 2020. Federated Heavy Hitters Discovery with Differential Privacy. In *International Conference on Artificial Intelligence and Statistics (AISTATS).* http://proceedings.mlr.press/v108/zhu20a.html http://proceedings.mlr.press/v108/zhu20a/zhu20a.pdf.

A GUMBEL MECHANISM

Definition 6 (Gumbel Mechanism $\mathcal{M}_{\mathcal{G}}$). *The Gumbel mechanism $\mathcal{M}_{\mathcal{G}}$, for utility function $u : (U^n \times \mathcal{R}) \to \mathbb{R}$ with sensitivity $\Delta u = \max_{\forall r \in \mathcal{R}, D \simeq D'} |u(D,r) - u(D',r)|$, outputs $r \in \mathcal{R}$ via*

$$\arg\max_{r \in \mathcal{R}} \{u(D,r) + \text{Gumbel}(2\Delta u/\epsilon)\},$$

where Gumbel(b) *denotes a random variable from the Gumbel distribution with scale b and density* $\text{Gumbel}(x;b) = \frac{1}{b}\exp\Big(-\Big(\frac{x}{b} + \exp\Big(-\frac{x}{b}\Big)\Big)\Big)$.

B PEMorig

PEMorig finds frequent prefixes of increasing lengths and split clients in $g = \lceil (b-\gamma)/\eta \rceil$ disjoint groups. The i-th group ($1 \le i \le g$) reports perturbed $(\gamma + i\eta)$-bit prefixes ($\gamma = \lceil \log_2 k \rceil$) of their datum to a server. In more detail, a user in group i selects a hash function $H : U \to \{1, \dots, u\}$ from a family of hash functions $\mathcal{H}$, where $u = \lceil \exp(\epsilon) + 1 \rceil$. Then, she applies *generalized randomized response* GRR over her hashed datum; more precisely, $h = \text{GRR}(H(d'))$ of the $(\gamma + i\eta)$-bit prefix d' of her datum d where

$$\text{GRR}(x) = \begin{cases} x & \text{with probability } p = \frac{\exp(\epsilon)}{\exp(\epsilon)+u-1} \\ y \neq x & \text{with probability } \frac{1}{\exp(\epsilon)+u-1} \end{cases},$$

and $y \in \{1, \dots, u\}$. Finally, she reports (H, h) to the server. Given the reports, the server creates a candidate set C by extending the previous top-2^γ prefixes with all possible binary strings of length η. Then, the server estimates the frequency of each prefix candidate $c \in C$ as $\frac{s_c - n/u}{p - 1/u}$ where s_c is the number of reports with matching hashes, i.e., $s_c = |\{c \mid c \in C \text{ and } H(c) = h\}|$.

C DISTRIBUTED GUMBEL NOISE

Random variable $X \sim \text{Gumbel}(b)$ can be expressed as

$$b \cdot \lim_{n \to \infty} \left\{ \sum_{j=1}^{n} \frac{Y_j}{j} - \log(n) \right\}, \quad Y_j \sim \text{Expon}(1),$$

where the Exponential distribution with scale b has density $\text{Expon}(x;b) = \frac{1}{b}\exp\Big(-\frac{x}{b}\Big)$ for $x > 0$ and 0 elsewhere [21].

While the Laplace distribution can be expressed as a *finite* sum, the Gumbel distribution requires an *infinite* sum. However, the expected approximation error for the Gumbel distribution can be made arbitrarily small in the number s of summands:

Theorem 9. *For* $\text{Gumbel}_{|s}(b) = b\sum_{j=1}^{s} \frac{Y_j}{j} - b\log(n)$, *where* $Y_j \sim \text{Expon}(1)$, *we have* expected approximation error $|\text{Gumbel}(b) - \text{Gumbel}_{|s}(b)| = O(b/s)$.

Proof. We have $\mathbb{E}[\text{Gumbel}(b)] = \gamma_{\text{EM}} \cdot b$, where $\gamma_{\text{EM}} \approx 0.5772$ is the Euler-Mascheroni constant, and $\mathbb{E}\big[\text{Gumbel}_{|s}(b)\big] = b\sum_{j=1}^{s} \frac{\mathbb{E}[Y_j]}{j} - b\log(n) \le b\big(\gamma_{\text{EM}} + 1/(2s) + O\big(1/s^2\big)\big)$, due to $\mathbb{E}[Y_i] = 1$ and [48, (4.30)]. Altogether,

$$\begin{aligned} &\big|\mathbb{E}\big[\text{Gumbel}(b) - \text{Gumbel}_{|s}(b)\big]\big| \\ &= \big|\mathbb{E}[\text{Gumbel}(b)] - \mathbb{E}\big[\text{Gumbel}_{|s}(b)\big]\big| \\ &\le |\gamma_{\text{EM}} b - b(\gamma_{\text{EM}} + O(1/s))| = bO(1/s). \end{aligned}$$

□

Table 2: Complexity of MPC protocols for b-bit integers (with pre-computed Beaver triples for AND, CondSwap) [3, 39].

Protocol	*Rounds*	*Interactive Operations*
ADD, NOT	0	0
Rec	1	1
AND, CondSwap	1	2
LE	4	$4b-2$
EQ	4	$b+4\log b$

Note that the input parties can pre-compute an arbitrary number of such sum terms, and add them to their prefix counts, thus, they only need to send a single message (i.e., the noisy counts) to the computation parties.

D COMPOSITION

Lemma 2. *Composing k approximate DP mechanisms requires less total ϵ budget compared to pure DP when $k > \frac{2\log(1/\delta')}{(2-\exp(\epsilon))^2}$ with $\delta' > 0$.*

Proof. Running k (ϵ, δ)-DP mechanisms on the same data leads to a total privacy budget of $k\epsilon$ for pure DP mechanisms ($\delta = 0$), and $(\sqrt{2k\log(1/\delta')}\epsilon + k\epsilon(\exp(\epsilon)-1), k\delta+\delta')$ for approximate DP mechanisms $(\delta, \delta' > 0)$ [37, Theorem 3.20]. Therefore, $\sqrt{2k\log(1/\delta')}\epsilon + k\epsilon(\exp(\epsilon)-1) < k\epsilon \Leftrightarrow \frac{1}{\sqrt{k}}\sqrt{2\log(1/\delta')} + (\exp(\epsilon)-1) < 1 \Leftrightarrow k > \frac{2\log(1/\delta')}{(2-\exp(\epsilon))^2}$. □

E COMPLEXITY OF MPC PROTOCOLS

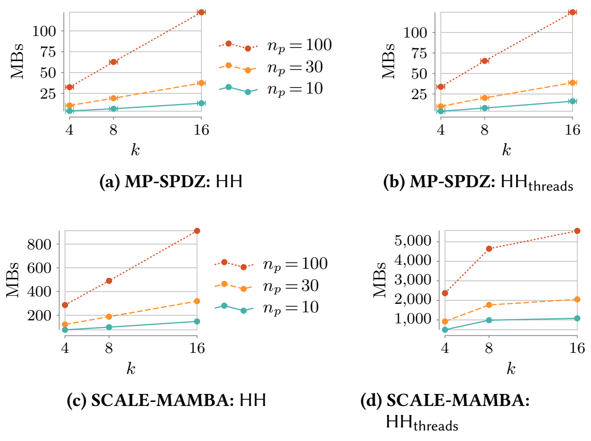
Table 2 lists the complexities for MPC protocols (without pre-computation for LE, EQ from [3]) typically measured in the number of *rounds* and *interactive operations*, where rounds describes the count of sequential interactive operations, and interactive operations (e.g., reconstruct sharing, multiplications) require each party to send messages to all other parties. Share reconstruction is denoted with Rec and $\text{NOT}(a) = 1 - a$.

Note that $\text{CondSwap}(a, b, c)$ is implemented with one multiplication and two additions $(b + (a - b) \cdot c)$ and $\text{AND}(a, b)$ also uses one multiplication $(a \cdot b)$. With pre-computed Beaver triples $\langle a\rangle, \langle b\rangle, \langle c\rangle$, where $c = ab$, multiplication $\langle x\rangle\langle y\rangle$ can be expressed as [13]: $\langle xy\rangle = \langle c\rangle + \alpha\langle b\rangle + \beta\langle a\rangle + \alpha \cdot \beta$, where $\alpha = \text{Rec}(\langle x - a\rangle)$, $\beta = \text{Rec}(\langle y - b\rangle)$.

F SECURE SORTING

We use the existing secure sorting based on merge sort from MP-SPDZ[7] and SCALE-MAMBA[8]. The implementations use conditional swaps: Roughly, whenever an array value $A[i]$ is smaller than $A[i+1]$, i.e., $b_{\text{swap}} = \text{LE}(A[i+1], A[i])$ is 1, they are swapped. However, we slightly adapt it, and re-use the comparison result b_{swap} to sort a second array B in the same way, i.e., for each swap with A we simply perform the same swap with B.

[7]https://github.com/data61/MP-SPDZ/blob/v0.1.8/Compiler/library.py#L464
[8]https://github.com/KULeuven-COSIC/SCALE-MAMBA/blob/862ecf547a01883cfbaf81a07c444c0c7cb53010/Compiler/library.py#L424



(a) **MP-SPDZ:** HH (b) **MP-SPDZ:** $\text{HH}_{\text{threads}}$ (c) **SCALE-MAMBA:** HH (d) **SCALE-MAMBA:** $\text{HH}_{\text{threads}}$

Figure 13: Communication per party for HH, $\text{HH}_{\text{threads}}$.

G AWS COSTS

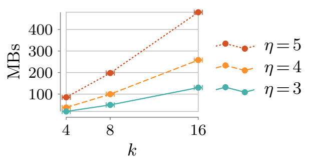
AWS t2.medium instances cost less than 5 Cents per hour, and communication of 1 GB costs around 2 Cents (per month) [6]. If one wants to optimize for cost, we suggest to use an MP-SPDZ implementation: All our MP-SPDZ evaluations for HH, PEM run in less than 30 minutes and require less than 1 GB of communication, hence, even our largest MP-SPDZ evaluation cost less than 5 Cents per computation party. (Except for $k = 16, \eta = 5$ which uses t2.large instances that costs less than 10 Cents per hour.) As a comparison, recall that LDP approach PEMorig requires up to 2^{20} hash computations for each user input. Our evaluation of PEMorig – also on t2.medium instances, without parallelization as this requires additional computational resources – showed running times of *hours* compared to the *minutes* required for PEM.

H COMMUNICATION OF HH & PEM

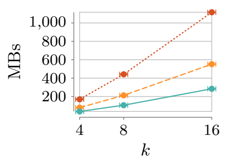
Figure 13 shows the communication per party for HH and $\text{HH}_{\text{threads}}$ and Figure 13 shows the communication for PEM. Overall, the sever communication for MP-SPDZ can be measured in MB whereas SCALE-MAMBA requires GB for larger evaluations.

I EVALUATION FOR LARGE DATA SIZES

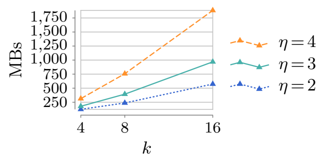
We designed our protocols with high accuracy on small data sizes n in mind, as it is the most challenging regime for DP where the noise easily exceeds the actual counts, particularly in a distributed setting. Nonetheless, our protocols also provide higher accuracy than local-model equivalents for large data sizes, e.g., 10^5, as visualized in Figure 15. We omitted the comparison to PEMorig with $\eta > 5$ as the evaluation did not finish after 12 hours on t2.medium instances. While PEMorig can be parallelized *per group* it is still linear in n and does not scale. Our protocol HH with fixed map size $t = 16$ is eventually outperformed by PEMorig for large enough data sets, i.e., around $n = 5 \cdot 10^5$. PEM, however, already finds almost all k heavy hitters for $n = 10^5$. These empirical observations confirm our analysis of HH and PEM detailed in Section 3.3 (i.e., PEM is better suited for larger data sets).



(a) **MP-SPDZ:** PEM, $b = 32$

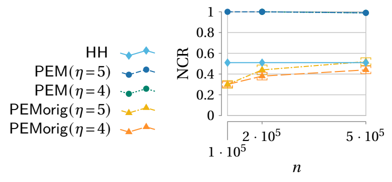


(b) **MP-SPDZ:** PEM, $b = 64$

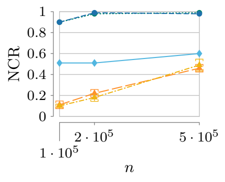


(c) **SCALE-MAMBA:** PEM, $b = 32$

Figure 14: Communication per party for PEM.



(a) Zipf



(b) Retail

Figure 15: NCR of PEM **variants and** HH **for fixed** $\epsilon = 0.25$, $k = 16$**, varying** $n \in \{10^5, 2 \cdot 10^5, 5 \cdot 10^5\}$.

J F1 SCORE

We also evaluated F1-scores (harmonic mean of precision and recall) and compare the relative difference of NCR to F1, i.e., (NCR−F1)/NCR. If NCR is 0, F1 is 0 as well, and we set the relative difference to 0. A positive value means NCR is larger than F1, which is to be expected. Recall, unlike F1, NCR gives more weight to elements that appear more frequently. However, negative values are possible (e.g., if the mode was not found).

Table 3 presents the relative difference of NCR to F1 averaged over $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$ for Zipf and retail data with $n = 1,000$. Table 4 presents the same for $n = 5,000$. Overall, the averaged scores for F1 and NCR are very close for our protocols (mostly the difference is below 6%) and further apart for PEMorig (mostly above 6% and up to 48% difference), i.e., our protocols provide superior F1 scores.

Table 5 gives the detailed comparisons for each ϵ on Zipf and retail data with $n = 1,000$ for fixed $k = 16$. Likewise, Table 6 presents the comparison for $n = 5,000$. Large relative differences for PEMorig result from its comparatively low scores. For example, PEMorig has NCR=0.1, F1=0.06 for $k = 16$, $n = 1,000$, $\epsilon = 2$ on retail data with a small absolute differences NCR−F1=0.04 leading to a large relative difference of 40% (last row in Table 5).

Data	k	HH	PEM ($\eta = 4$)	PEM ($\eta = 5$)	PEMorig
Zipf	4	2.6%	1.6%	2.0%	14.8%
	8	4.7%	1.4%	2.3%	16.7%
	16	4.0%	0.6%	2.0%	20.0%
Retail	4	1.1%	-2.1%	-0.5%	6.1%
	8	3.6%	-0.6%	1.5%	9.2%
	16	4.1%	0.0%	1.2%	48.0%

Table 3: Relative comparison (NCR–F1)/NCR, varying $k \in \{4, 8, 16\}$ **for** $n = 1,000$ **averaged over** $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$.

Data	k	HH	PEM ($\eta = 4$)	PEM ($\eta = 5$)	PEMorig
Zipf	4	0.9%	2.0%	1.4%	25.0%
	8	7.0%	3.4%	5.6%	25.1%
	16	5.7%	5.2%	5.0%	47.1%
Retail	4	1.5%	5.3%	10.9%	3.6%
	8	4.9%	3.5%	1.0%	15.8%
	16	5.0%	7.2%	5.4%	8.1%

Table 4: Relative comparison (NCR–F1)/NCR, varying $k \in \{4, 8, 16\}$ **for** $n = 5,000$ **averaged over** $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$.

Data	ϵ	HH	PEM ($\eta = 4$)	PEM ($\eta = 5$)	PEMorig
Zipf	0.1	0.0%	0.0%	0.0%	0.0%
	0.25	4.3%	0.0%	0.0%	0.0%
	0.5	3.0%	0.0%	0.0%	0.0%
	1	5.9%	0.0%	4.3%	50.0%
	2	7.0%	3.0%	5.9%	50.0%
Retail	0.1	0.0%	0.0%	0.0%	0.0%
	0.25	0.0%	0.0%	0.0%	0.0%
	0.5	2.9%	0.0%	0.0%	100.0%
	1	9.0%	0.0%	0.0%	100.0%
	2	8.6%	0.0%	6.0%	40.0%

Table 5: Relative comparison (NCR–F1)/NCR, varying $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$ **for** $k = 16, n = 1,000$.

K MPC FRAMEWORKS

We deployed SCALE-MAMBA [4] version 1.3 and MP-SPDZ [53] version 0.1.8 in our evaluation. Here, we evaluated HH, $\text{HH}_{\text{threads}}$ without the final sorting step.

SCALE-MAMBA: Version 1.3 vs. 1.9: Out-of-the-box, i.e., without adjusting options and runtime switches, SCALE-MAMBA version 1.3 was faster than (at time of evaluation most recent release) 1.9 for our protocols. Versions 1.4 to 1.9 mainly added features which our protocols do not rely on (e.g., support for Garbled Circuits, authenticated bits). We used runtime switch -dOT from version 1.9, to reduce offline data creation (for features we are not using), for a

Data	ϵ	HH	PEM ($\eta = 4$)	PEM ($\eta = 5$)	PEMorig
Zipf	0.1	3.0%	0.0%	0.0%	0.0%
	0.25	7.0%	4.3%	3.8%	100.0%
	0.5	7.0%	5.7%	5.3%	45.5%
	1	5.9%	8.2%	8.7%	46.7%
	2	5.9%	7.5%	7.4%	43.3%
Retail	0.1	0.0%	0.0%	0.0%	0.0%
	0.25	3.0%	6.3%	3.6%	0.0%
	0.5	5.4%	8.1%	7.2%	0.0%
	1	8.3%	7.1%	7.6%	0.0%
	2	8.2%	14.8%	8.3%	40.6%

Table 6: Relative comparison (NCR−F1)/NCR, varying $\epsilon \in \{0.1, 0.25, 0.5, 1, 2\}$ for $k = 16, n = 5,000$.

fairer comparison with 1.3. Still, in our brief evaluation, we found 1.9 to be somewhat slower:

- For PEM with $k = 8, \eta = 2, b = 32$ runtime increased by around 20% from 1.3 to 1.9 (≈206 vs. 248 s).
  Without `-dOT` communication almost doubled (≈237 vs. 460 MB), with `-dOT` it remained about the same.
- For HH$_{\text{threads}}$ with $k = 16$ runtime increased by around 10% from 1.3 to 1.9 (≈600 vs. 667 s).
  Without `-dOT` communication increased by around 30% (≈5.5 vs. 7.2 GB) with `-dOT` it remained about the same.

MP-SPDZ: Semi-honest vs. Malicious: MP-SPDZ supports semi-honest as well as malicious security for multiple secure computation paradigms (e.g,. Shamir secret sharing, BMR) [53], whereas SCALE-MAMBA only supports malicious security. In Section 5 we evaluated semi-honest MP-SPDZ. Next, we briefly compare SCALE-MAMBA and MP-SPDZ for *maliciously secure Shamir*:

- For PEM with $k = 16, \eta = 4, b = 32$, MP-SPDZ is more than twice as fast than SCALE-MAMBA (≈14 vs. 30 minutes) with around 400 MB less communication (≈1.47 vs. 1.88 GB).
- For HH with $k = 16$ and $n_p = 30$ per party $p \in \{1, 2, 3\}$, MP-SPDZ is roughly 27% slower than SCALE-MAMBA (≈6 vs. 4.7 minutes), but requires around 60% less communication (≈192 vs. 313 MB).

This suggests that, for malicious security and considering only running time, PEM is more efficient with MP-SPDZ, whereas HH is more efficient with SCALE-MAMBA.