Hash-Prune-Invert: Improved Differentially Private Heavy-Hitter Detection in the Two-Server Model

Borja Balle*, James Bell†, Albert Cheu†, Adria Gascon†,
Jonathan Katz‡, Mariana Raykova‡, Phillipp Schoppmann‡, and Thomas Steinke*
*Google DeepMind
†Google Research
‡Google

***Abstract*—Differentially private (DP) heavy-hitter detection is an important primitive for data analysis. Given a threshold t and a dataset of n items from a domain of size d, such detection algorithms ignore items occurring fewer than t times while identifying items occurring more than $t+\Delta$ times; we call Δ the *error margin*. In the central model where a curator holds the entire dataset, (ε, δ)-DP algorithms can achieve error margin $\Theta(\frac{1}{\varepsilon}\log\frac{1}{\delta})$, which is optimal when $d \gg 1/\delta$.**

Several works, e.g., Poplar (S&P 2021), have proposed protocols in which two or more non-colluding servers jointly compute the heavy hitters from inputs held by n clients. Unfortunately, existing protocols suffer from an undesirable dependence on $\log d$ in terms of both server efficiency (computation, communication, and round complexity) and accuracy (i.e., error margin), making them unsuitable for large domains (e.g., when items are kB-long strings, $\log d \approx 10^4$).

We present *hash-prune-invert* (HPI), a technique for compiling any heavy-hitter protocol with the $\log d$ dependencies mentioned above into a new protocol with improvements across the board: computation, communication, and round complexity depend (roughly) on $\log n$ rather than $\log d$, and the error margin is independent of d. Our transformation preserves privacy against an active adversary corrupting at most one of the servers and any number of clients. We apply HPI to an improved version of Poplar, also introduced in this work, that improves Poplar's error margin by roughly a factor of $\sqrt{n}$ (regardless of d). Our experiments confirm that the resulting protocol improves efficiency and accuracy for large d.

1. Introduction

Distributed *heavy-hitter detection* is a ubiquitous analytics task. Here, given n clients each holding an item from a large domain of size d, the goal is to identify items held by at least some threshold t of the clients. In many settings—e.g., identifying URLs associated with browser crashes—maintaining privacy of the item held by any particular client is paramount. In those same applications, the domain size d can be massive (e.g., $\log d \approx 10^4$), as it depends on the space of possible URLs, the types of errors that may occur, and other metadata. (See, e.g., Mastic [1] for a description of Network Error Logging [2].)

A line of work [1], [3], [4], [5], [6], [7] has looked at preserving client privacy by assuming the clients share their data among two or more non-colluding servers, which then interact to compute the heavy hitters. Two of these solutions, Poplar [4] and Prio [3], are currently part of a standardization effort [8] by the Crypto Forum Research Group (CFRG), part of the Internet Engineering Task Force (IETF); they have also been adopted in a number of real-world settings [9].

Unfortunately, Poplar and its subsequent extensions offer weak privacy guarantees if one of the servers is malicious, since a malicious server can bias the counts used to identify heavy hitters and, as we illustrate in Section 3.1.3, these biases may completely expose a client holding an outlier input. Both the Poplar authors and the CFRG [8, Section 9.4.1] explicitly recognize *differential privacy (DP)* as a way to defend against such attacks and to offer additional privacy protection for clients.

DP uses noise to perturb an algorithm's output. As a consequence, it is not possible for a DP heavy-hitter detection algorithm A to always output all items held by t or more users while never outputting items held by fewer than t users; rather, A must have some *error margin* Δ. Roughly, this means A guarantees that (1) any item held by fewer than t users is output with low probability, while (2) any item held by more than $t + \Delta$ users is output with high probability. Items held by at least t users but at most $t + \Delta$ users may or may not be output.

Boneh et al. [4, Appendix E] sketch an approach to perform DP heavy-hitter detection based on Poplar; we call the resulting protocol DP-Poplar. The error margin of DP-Poplar grows roughly as $\sqrt{n \log d}$, which is significantly worse than what can be achieved by DP algorithms in the centralized setting where $\Delta = O(\frac{1}{\varepsilon}\log\frac{1}{\delta})$ is achievable in the typical case when $\delta = O(1/n)$ [10], [11], [12]. Moreover, DP-Poplar has an undesirable dependence on $\log d$ (inherited from Poplar) in server computation, communication, and round complexity. These drawbacks make DP-Poplar impractical for very large d.

This motivates the following question:

Is there a two-server protocol for actively secure DP heavy-hitter detection whose server efficiency and error margin (per-heavy hitter) is independent of the size of the input domain?

1.1. Our Contributions

We show a positive answer to the above question. First, and of independent interest even for small d, we show an improvement to DP-Poplar that reduces the error margin by roughly a factor of $\sqrt{n}$. Second, and what we consider our main contribution, we show how to use *hashing* to further improve accuracy while also improving server efficiency. While hashing is a natural approach to handle large domains—and was also suggested for Poplar [4, App. B]—it is difficult to use hashing while (1) ensuring the ability to recover the heavy hitters themselves (and not just their hashes), and (2) maintaining differential privacy with error margin independent of d. In particular, the hashing-based solution described for Poplar applies only when computing exact heavy hitters, and does *not* apply to DP-Poplar.

In more detail, our contributions include the following:

- GaussTrie, an improvement to DP-Poplar that removes a $\sqrt{n}$ term from its error margin. See Sections 3.2.1 and 4.1 for further details.
- *Hash-prune-invert* (HPI), a generic framework for compiling any DP heavy-hitter detection protocol to improve its efficiency and accuracy in the large-domain setting ($\log d \gg \log n$). At a high level, we run the original protocol on *hashed inputs* to obtain a set of frequent hashes, and then invert those hashes to recover the heavy hitters themselves.[1] A simple way to invert a hash h would be to output the preimage of h in the input dataset with the largest frequency; this, however, would not be differentially private. We address this by using *pruning* to remove a hash if a DP test indicates that its inversion is sensitive. Pruning ensures that inverting the surviving hashes does not leak information. Section 3.2 gives a more detailed overview of our HPI framework, and full details are in Section 4.3.
- An efficient secure two-party computation (2PC) protocol implementing HPI+GaussTrie (Section 5). The protocol ensures privacy against any number of malicious clients colluding with a single malicious server. Our key contributions here are an efficient approach for distributed hash inversion along with carefully designed boolean circuits for noise generation. We empirically find that these steps add little overhead beyond what is already needed in Poplar.
- We evaluate GaussTrie and HPI+GaussTrie on synthetic datasets (Section 6.2) to assess recall, F1 score, and efficiency. Consistent with theory, we find that hashing improves accuracy and efficiency.

1. Privacy holds even if there are hash collisions, but for good accuracy we require collisions to occur with low probability.

We summarize the performance of our 2PC protocols for GaussTrie and HPI+GaussTrie in Table 1. Our two-server protocol improves upon Poplar in terms of both error margin and server efficiency. Replacing $\log d$ factors by $\log n$ factors allows for scaling to applications where $\log d \gg \log n$.

1.2. Other Related Work

Besides relying on two or more non-colluding servers, other approaches to private heavy-hitter detection have been considered. Some works [13], [14] have looked at DP heavy-hitter detection in the local model where the clients add noise to their own inputs before sending them to a curator. Those works have error margin that scales with $\sqrt{n}$ and $\log d$. In the *shuffle model*, which augments the local model with a primitive for anonymizing messages, it is possible to achieve error margin $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ [15]. However, the only known way to achieve that bound is with a protocol that requires each client to send $\Omega(d)$ messages in expectation. Moreover, the most common implementation of the shuffle model already requires an additional trusted party to shuffle messages before forwarding them to the curator.

Bell et al. [16] use secure 2PC to implement the (optimal) centralized DP heavy-hitter algorithm [10], [11], [12]. Their protocol is only secure for semi-honest adversaries. In subsequent work, Braun et al. [17] show how to make their protocol actively secure. Both solutions are less efficient than Poplar, and are much less efficient than our protocol for large d.

Durak et al. [18] propose Precio, a three-server histogram estimation protocol. Although there are similarities between their work and ours, they assume a semi-honest adversary and work in the honest-majority setting (i.e., at most one server can be corrupted). Their experiments consider data from domains of size $\leq 2^{22}$, which is significantly smaller than the regime we target.

2. Definitions and Notation

2.1. Setting and Threat Model

We consider n clients and two servers executing some protocol Π. We assume a static adversary $\mathcal{A}$ who may corrupt any number of clients and at most one of the servers. We let $x_i \in \mathcal{D} = [d]$ denote the input of the ith client, and let X be the multiset of honest clients' inputs. We let λ be a computational security parameter given as input to the clients, the servers, and the adversary. We write $\text{REAL}_{\Pi,\mathcal{A}}(1^\lambda, X)$ for the random variable corresponding to the view of $\mathcal{A}$ and the output of the honest clients in an execution of Π when honest clients have inputs given by X.

		DP-Poplar [4, Appendix E]	GaussTrie	HPI+GaussTrie
Error margin		$\sqrt{\frac{n}{t}\log d \log n \log(\frac{n}{t}\log d)}$	$\sqrt{\log d \log n \log(\mathsf{Heavy}_t \log d)}$	$\log n \cdot \sqrt{\log(\mathsf{Heavy}_t \log n)}$
Servers	Computation	$\mathsf{Heavy}_t \cdot n \cdot \log d$	$\mathsf{Heavy}_t \cdot n \cdot \log d$	$\mathsf{Heavy}_t \cdot (n \cdot \log n + \log d)$
	Communication (bits)	$\mathsf{Heavy}_t \cdot \log n \cdot \log d$	$\mathsf{Heavy}_t \cdot \log n \cdot \log d$	$\mathsf{Heavy}_t \cdot (\log^2 n + \log d)$
	Rounds	$\log d$	$\log d$	$\log n$

TABLE 1: DP heavy-hitter detection protocols in the 2-server model. All costs are asymptotic, and omit big-O notation as well as dependencies on ε, δ for clarity. We let n denote the number of clients (i.e., the number of inputs), let $d \gg n$ denote the domain size, and let Heavy_t denote the number of heavy hitters. We assume $\delta \ll 1/n$.

We also define an ideal world that allows us to characterize the security properties of Π. Here we have a trusted entity evaluating an ideal functionality $\mathcal{F}$ for n clients in the presence of an adversary $\mathcal{A}$ (also called a *simulator*) who may again statically corrupt any number of clients and at most one server. Honest clients send their inputs to $\mathcal{F}$, and $\mathcal{A}$ sends inputs to $\mathcal{F}$ on behalf of the corrupted clients. We again let X denote the honest clients' inputs, and let $\hat{X}$ denote the multiset of malicious clients' inputs; the total input is $\bar{X} := X \cup \hat{X}$. The computation of $\mathcal{F}$ may be interactive, and thus involve sending intermediate values to $\mathcal{A}$ and receiving additional inputs (whose purpose depends on the specifics of $\mathcal{F}$) from $\mathcal{A}$. In particular, $\mathcal{A}$ can send `ABORT` to $\mathcal{F}$ and prevent honest clients from receiving the output. Following the computation, $\mathcal{A}$ may output an arbitrary function of its view. We let $\text{IDEAL}_{\mathcal{F},\mathcal{A}}(1^\lambda, X)$ be the random variable corresponding to the output of $\mathcal{A}$ and the output of the honest clients when honest clients have inputs given by X.

Definition 1 (Secure computation)**.** Π *securely computes* $\mathcal{F}$ if for every polynomial-time real-world adversary $\mathcal{A}$ there is a polynomial-time ideal-world adversary $\mathcal{A}'$ such that for all X, the distributions $\text{REAL}_{\Pi,\mathcal{A}}(1^\lambda, X)$ and $\text{IDEAL}_{\mathcal{F},\mathcal{A}'}(1^\lambda, X)$ are computationally indistinguishable.

2.2. Differential Privacy (DP)

For random variables V, V', we write $V \approx_{\varepsilon,\delta} V'$ if the following holds for any S:

$$\Pr[V \in S] \leq \exp(\varepsilon) \cdot \Pr[V' \in S] + \delta$$
$$\Pr[V' \in S] \leq \exp(\varepsilon) \cdot \Pr[V \in S] + \delta.$$

We write $V \approx_\varepsilon V'$ for $V \approx_{\varepsilon,0} V'$. Two equal-sized multisets $X, X' \in \mathcal{D}^*$ are *neighboring* if they differ in exactly one element, and we write $X \sim X'$ in that case.

Definition 2. An ideal functionality $\mathcal{F}$ satisfies (ε, δ)-differential privacy ((ε, δ)-DP) if, for all $\mathcal{A}$, λ, and $X \sim X'$,

$$\text{IDEAL}_{\mathcal{F},\mathcal{A}}(1^\lambda, X) \approx_{\varepsilon,\delta} \text{IDEAL}_{\mathcal{F},\mathcal{A}}(1^\lambda, X').$$

We define a notion of computational differential privacy for protocols by adapting the definition of SIM$^+$-CDP [19].

Definition 3. A protocol Π satisfies (ε, δ)-computational differential privacy ((ε, δ)-CDP) if there is an (ε, δ)-DP ideal functionality $\mathcal{F}$ such that Π securely computes $\mathcal{F}$.

2.3. Heavy-Hitter Detection

Roughly, the goal of heavy-hitter detection is to identify the elements $\mathsf{Heavy}_t(\bar{X})$ that occur $\geq t$ times in the input $\bar{X}$, for some $t \geq 1$. When DP is required, however, it is not possible to have a sharp threshold t.[2] Instead there must be some *error margin* Δ around t; Δ may depend on $\bar{X}$.

Definition 4. Ideal functionality $\mathcal{F}$ performs *t-heavy-hitter detection over $\mathcal{D}$ with error margin $\Delta(\cdot)$ and error rate β* if (when $\mathcal{A}$ is passive) given any $\bar{X}$ over $\mathcal{D}$, with probability at least $1 - \beta$ it outputs a set $S \subseteq \mathcal{D}$ such that

- all elements that occur fewer than t times in $\bar{X}$ *are not* included in S
- all elements that occur more than $t + \Delta(|\bar{X}|)$ times in $\bar{X}$ *are* included in S.

Our work focuses on identifying heavy hitters, not reporting estimates of their frequencies ("histogram estimation"). In Appendix A, however, we show that such estimates are straightforward to obtain once detection is performed. As a corollary, lower bounds on the error of DP histogram estimation imply lower bounds on the error margin of DP heavy-hitter detection.

Remark 5 (Implications for recall & F1 score)**.** *Suppose $\mathcal{F}_1, \mathcal{F}_2$ both perform t-heavy hitter detection but the error margins satisfy $\Delta_1 < \Delta_2$. Then for any dataset, the recall & F1 score*[3] *for $\mathcal{F}_1$ are at least that of $\mathcal{F}_2$. This is because $\mathcal{F}_1$ will find any item with count $\in [t + \Delta_1, t + \Delta_2]$ while $\mathcal{F}_2$ lacks that property.*

In the case where $\mathcal{F}_1$ performs t-heavy hitter detection but $\mathcal{F}_2$ performs $t - \gamma$-heavy-hitter detection with the same margin, the impact on F1 score (defined w.r.t. t) is data-dependent.

2. If we were to report a set S containing exactly the inputs with count $\geq t$ then one client can wholly determine membership in S.

3. The F1 score on a given dataset is equal to $2\mathsf{TP}/(2\mathsf{TP} + \mathsf{FP} + \mathsf{FN})$, where TP is the number of true positives, and FP (resp., FN) is the number of false positives (resp., false negatives). "Positive" (resp. "negative") corresponds to elements with frequency at least (resp. below) t.

3. Technical Overview

We provide a high-level overview of our protocol for differentially private heavy-hitter detection. We begin by reviewing prior work on which our solution is based.

3.1. Trie-Based Heavy-Hitter Detection

Recall that in our setting there are n users with the ith user holding $x_i \in [d]$; and we denote by $X = \{x_i\}$ the multiset of users' inputs (assuming all users are honest for simplicity). In the exact version of the heavy-hitters problem (without differential privacy), the goal is to securely identify heavy hitters, i.e., items held by at least some public threshold t of the users, using two non-colluding servers.

One natural solution to this problem is to have each user generate a 1-hot encoding $\vec{v}_i$ of its input—that is, $\vec{v}_i$ will be a length-d binary vector with a single 1 at the index corresponding to x_i—and then secret share $\vec{v}_i$ with the two servers. Each server can locally sum its shares, and then the servers can run a lightweight 2PC protocol to identify the indices whose count is at least t. This becomes useless when items are even moderately long since client-to-server and server-to-server communication are both $\Theta(d)$.

3.1.1. Using DPFs. Distributed point functions (DPFs) [20], [21], [22] can be used to reduce the client-to-server communication in the above approach. A DPF allows the ith user to generate (short) keys $\mathsf{key}_{i,0}, \mathsf{key}_{i,1}$ such that for all $x \in \{0,1\}^{\log d}$ it holds that

$$\mathsf{Eval}_{\mathsf{key}_{i,0}}(x) + \mathsf{Eval}_{\mathsf{key}_{i,1}}(x) = \begin{cases} 1 & x = x_i \\ 0 & \text{otherwise.} \end{cases}$$

On the other hand, each key on its own reveals nothing about x_i. By sending $\mathsf{key}_{i,j}$ to server j, the user can share its 1-hot vector $\vec{v}_i$ with the servers as before, but with much less communication. While this improves the client-to-server communication, the server computation and server-to-server communication remain $\Theta(d)$.

3.1.2. Trie-based Solutions and Poplar. Trie-based solutions [23], [24], [25], [26] can be used to compute heavy hitters more efficiently than the approach above, leading to solutions that are feasible even for large domains. Consider the central setting for the moment, where a single server holds X. If elements in the domain are represented as strings over $[b]$ of length h (so $d \leq b^h$), then a trie-based algorithm constructs an incomplete tree of branching factor $\leq b$ and height $\leq h$ such that the node corresponding to a prefix p is extended iff the count c_p of the items in X with prefix p exceeds some (possibly variable) threshold. Leaves will correspond to heavy hitters. The overhead of this approach scales as $O(\mathsf{Heavy}_t)$, where Heavy_t is the number of heavy hitters in X, regardless of the size of the domain.

In the Poplar work [4], Boneh et al. observe that trie-based algorithms can also be efficiently computed in the distributed setting using DPFs. The main idea is for the ith client to generate keys $\mathsf{key}_{i,0}, \mathsf{key}_{i,1}$ so that for all $p \in [b]^{\leq h}$ it holds that

$$\mathsf{Eval}_{\mathsf{key}_{i,0}}(p) + \mathsf{Eval}_{\mathsf{key}_{i,1}}(p) = \begin{cases} 1 & p \text{ is a prefix of } x_i \\ 0 & \text{otherwise.} \end{cases}$$

(While this can be done by having each user generate h independent DPF keys, Boneh et al. show how it can be done more efficiently using *incremental* DPFs.) As in the approach described earlier, the servers can use these keys to securely check whether c_p exceeds some threshold for any desired prefix p, and thus securely emulate a trie-based algorithm. Assuming $b = O(1)$, the resulting 2PC protocol requires $O(h) = O(\log d)$ rounds of communication between the servers and $O(\mathsf{Heavy}_t)$ server-to-server communication.

3.1.3. Adding Differential Privacy. Unfortunately, Poplar and subsequent improvements [1], [7] do not provide good privacy guarantees in the presence of an actively corrupted server. To illustrate this, we describe a simple adversary who learns any input(s) held by exactly one user. This is clearly inconsistent with an intuitive notion of privacy by which the output should only reveal inputs whose multiplicity is at least t. (We remark that, irrespective of this attack, differential privacy provides stronger guarantees than securely computing the set of exact heavy hitters. The purpose of describing this attack is to show that privacy issues for Poplar-like protocols are even more significant.)

The root of the problem is that a corrupted server can shift counts while exploring the trie described above. Concretely, assume server 1 is corrupted. While it is expected to output $c_{p,1} := \sum_i \mathsf{Eval}_{\mathsf{key}_{i,1}}(p)$ for paths p in the trie, it can output $c_{p,1} + \Delta_p$ for Δ_p an arbitrary offset the adversary's choice. The observed count $c_{p,0} + c_{p,1}$ is then shifted by Δ_p from the true count. Our attacker sets $\Delta_p = t - 1$ for all prefixes p starting from the root. As soon as a prefix p with observed count equal to t is observed, the attacker knows it has identified a prefix that corresponds to an input x held by a single user. To completely recover x the attacker sets $\Delta_{p'} = t - 1$ for all prefixes p' that extend p.

The above attack does not contradict the security claims of Poplar; in fact, the authors of Poplar themselves note that differential privacy is needed if there are not "good statistical guarantees on the entropy of the inputs contributed by honest clients." Motivated by this, Boneh et al. propose a differentially private variant of Poplar that we call DP-Poplar [4, Appendix E]. The main idea is to have each server add Laplace noise (calibrated to provide ε-DP) to its share of c_p, so the observed counts are noisy rather than exact.

3.2. Our Solution

As discussed already in the introduction, the main drawbacks of DP-Poplar are that it has relatively large error margin proportional to $\sqrt{n \log d}$, and server efficiency depending on $\log d$ (cf. Table 1). We describe here how we address these issues.

3.2.1. GaussTrie. Our first contribution is a variant of DP-Poplar called GaussTrie, which incorporates three differences in the underlying trie-based algorithm. First, we replace Laplace noise with Gaussian noise to facilitate tighter privacy accounting. Second, we improve the sensitivity bound of Boneh et al. Together, these changes eliminate a factor of $\sqrt{n}$ from the error margin (cf. Table 1). As an additional modification, we give the option to negatively bias the noisy counts at each level (by a public amount); this has the effect of reducing the number of nodes explored, trading off accuracy for efficiency (which is a worthwhile tradeoff when the trie-based algorithm is emulated by a 2PC protocol as it will be in the two-server setting we consider).

Even with these improvements, GaussTrie's error margin still depends on $\log d$, and server efficiency is unchanged as compared to DP-Poplar (cf. Table 1).

3.2.2. The Promise and Challenges of Hashing. A natural idea to avoid dependence on d is to use *hashing* to reduce the size of the domain, i.e., each client will hash its input and then the parties will run a private heavy-hitter detection algorithm (e.g., GaussTrie) on the hashed results. Assuming no collisions, this results in a set S_{init} containing hashes of heavy hitters. The elements in S_{init} can then be *inverted* to recover the heavy hitters themselves.

It is crucial, however, for the hash inversion to be done in a way that preserves differential privacy. To see the subtleties that can arise, consider the following protocol: each client shares $(x_i, H(x_i))$ with the servers, which (1) run a private heavy-hitter detection protocol on the hashes to compute S_{init} and then (2) use generic 2PC to find elements of $X = \{x_i\}$ that are preimages of elements of S_{init}. Malicious clients can violate privacy of an honest client by sending malformed values to the servers.[4] Specifically, fix neighboring datasets X, X' of honest clients' inputs, where some value x is held by one honest client in X but by no honest clients in X'. Malicious clients can share inputs of the form $(\perp, H(x))$ with the servers to ensure that $H(x)$ ends up in S_{init} regardless of whether honest clients use X or X'. But inversion of $H(x)$ will succeed iff dataset X was used.

The above discussion highlights the fact that inversion relies on the input dataset. However, while a given dataset may have the information required to invert $H(x)$, a neighboring dataset might not. Malicious clients complicate the situation, as they might misbehave and contribute to the protocol a pair (x, h) where $h \neq H(x)$.

4. While such an attack could potentially be prevented by having clients give ZK proofs of correctness, this imposes significant additional work on the clients. In general, we view clients as being much more computationally limited than the servers.

3.2.3. Intuition for HPI. We solve the aforementioned issues using a technique we call Hash-Prune-Invert (HPI). Our key idea is to *prune* certain hashes from S_{init} before performing the inversion step so we can avoid inverting hashes that might violate privacy. Details follow.

We start by describing concretely how inversion is done. To compute a preimage of a given hash $s \in S_{\text{init}}$, the servers run a series of "elections": for $j \in \{1, \ldots, \log d\}$, the servers compute $\mathsf{count}[j, 0]$, the number of preimages of s in X with jth bit equal to 0, and $\mathsf{count}[j, 1]$, the number of preimages of s in X with jth bit equal to 1. Let $\mathsf{gap}_j = \mathsf{count}[j, 0] - \mathsf{count}[j, 1]$. The protocol releases 0 as the value of the jth bit of the preimage if $\mathsf{gap}_j \geq 0$, and 1 otherwise. It is now even more clear that differential privacy is not guaranteed by inversion because, e.g., changing the input of a single client may change the outcome of one of the elections.

At the heart of our contribution is the observation that we can *prune* hash values in S_{init} if their computed inverse is "unstable" to a client substitution, that is, if there is an index j such that $|\mathsf{gap}_j|$ is "too small." A naive attempt to prune in a differentially private way is to add noise to each of the $2 \log d$ counts and then post-process the results. This would cause privacy loss to accumulate across all the counts, and the error margin would still scale with $\log d$. A crucial observation is that we can incur just $O(1)$ privacy loss here, since we do not need to add noise to all the vote totals; instead, we consider the *minimum margin of victory* $\mathsf{gap} = \min\{|\mathsf{gap}_j|\}$. If gap is sufficiently large, then no single client's input can change any election outcome. Thus it suffices to obtain a DP estimate of gap whose magnitude we can compare against a threshold. Since gap itself can change by at most 1 in neighboring datasets, a DP estimate of gap requires only $O(1)$ noise. It is also worth noting that malicious inputs do not change the sensitivity of gap.

Concretely, then, we prune (i.e., refuse to invert) a particular hash s if $\mathsf{gap} + \eta$ is too small, where $\eta \leftarrow \mathtt{Laplace}(1/\epsilon)$. (This pruning step is performed *obliviously*, i.e., by evaluating $\mathsf{gap} + \eta$ and comparing the result to a public threshold using a 2PC protocol, which we discuss in Section 5.2.) This allows us to bound the probability of the bad event that was central to the attack on Poplar described in Section 3.1.3. More generally, it ensures that the protocol's behavior on neighboring datasets does not change too much.

Table 1 shows the resulting costs when applying HPI to GaussTrie. Choosing the hash function to have a range of size $O(n^2)$ suffices to avoid collisions with sufficiently high probability, and allows performance and error margin to scale[5] with $O(\log n)$ rather than $O(\log d)$.

3.2.4. DPF-based HPI. As discussed above, the blueprint for our protocol is to first identify heavy hashes and then invert them privately. To facilitate this, each client i will secret share with the servers information about its input x_i and the hash $h_i := H(x_i)$. More concretely, each client generates *incremental* DPF keys $\mathsf{key}_{i,0}, \mathsf{key}_{i,1}$ such that

$$\sum_{b\in\{0,1\}} \mathsf{Eval}_{\mathsf{key}_{i,b}}(p) = \begin{cases} 1 & p \sqsubset h_i \\ 1\|\mathtt{BinHist}(x_i) & p = h_i \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where $p \sqsubset h_i$ means that p is a strict prefix of h_i, and $\mathtt{BinHist}(x_i)$ is a $2\log_2 d$-bit encoding of x_i obtained from its binary encoding by replacing '1' with '01' and '0' with '10'. (For example, $\mathtt{BinHist}(12) = 01011010$ since the binary encoding of 12 is 1100.) This allows the servers to construct a trie over the hash values, with the leaves corresponding to heavy hashes along with an encoding of their inverses (as reported by clients). The encoding, in turn, allows the servers to efficiently compute the counts needed to determine whether to prune or not and, if not, to perform inversion. We emphasize in particular that our approach avoids the need to evaluate the hash function H using 2PC. As we show in Section 5, there is instead a 2PC protocol for evaluating HPI that uses just a small number of secure comparisons and additions.

4. Ideal Functionalities: GaussTrie & HPI

In this section, we describe ideal functionalities for our HPI framework and the underlying GaussTrie algorithm with which we instantiate that framework. Our functionalities explicitly incorporate interactions with an adversary, indicated in the pseudocode by the symbol ▶. Our accuracy and privacy analyses carry over to protocols that securely compute these functionalities in a two-server setting.

Our description in this section is "bottom-up": We begin in Section 4.1 by formalizing GaussTrie, the heavy-hitter detection algorithm we run on (hashed) user data. We introduce a subroutine for pruning and inverting the output of that algorithm in Section 4.2. We put everything together in Section 4.3.

4.1. GaussTrie

Given a multiset $\bar{X}$ containing strings in $[b]^h$, the GaussTrie algorithm constructs a trie of branching factor $\leq b$ and height $\leq h$. (See Ideal Functionality 1.) Nodes in the trie correspond to prefixes in the natural way (with the root node corresponding to the empty string), and the main idea of the algorithm is to iteratively extend a node corresponding to a string p only if the number of elements in $\bar{X}$ having prefix p is sufficiently large (cf. line 23). To preserve differential privacy, the algorithm cannot make this decision based on the exact count c_p of the number of elements in $\bar{X}$ with prefix p; instead, a noisy count $\tilde{c}_p$ is used (cf. line 21). As noted earlier (and in contrast to Poplar), GaussTrie uses Gaussian noise[6] to compute $\tilde{c}_p$.

Another difference from Poplar is that GaussTrie can be configured to negatively bias the noisy counts before comparing to a threshold. We introduce this bias to reduce the probability of exploring prefixes that occur fewer than t times. Increasing the magnitude of the bias makes nodes less likely to be extended, and so decreases the false-positive rate but increases the false-negative rate. More importantly, it improves efficiency (since fewer nodes are extended), something that is critical when we securely realize the functionality by a two-server protocol. We stress that privacy of GaussTrie holds whether or not biases are used.

The functionality gives the noisy counts of the explored nodes to the adversary $\mathcal{A}$. Those noisy counts incorporate noise from the functionality itself, as well as an arbitrary contribution from $\mathcal{A}$. The adversary's contribution may affect correctness, but will not affect privacy since noise is also added by the functionality. We allow $\mathcal{A}$ to simply abort the functionality at any point. A semi-honest (or passive) adversary chooses its noise contributions from the same distribution $N(0, \sigma^2)$ as the noise chosen by the functionality.

We now state the privacy guarantee of GaussTrie.

Theorem 6 (Privacy of GaussTrie)**.** *Let σ satisfy*

$$\Phi\left(\frac{\sqrt{h}}{2\sigma} - \frac{\varepsilon\sigma}{\sqrt{h}}\right) - e^{\varepsilon} \cdot \Phi\left(-\frac{\sqrt{h}}{2\sigma} - \frac{\varepsilon\sigma}{\sqrt{h}}\right) \leq \delta.$$

(For $\varepsilon < 1$, taking $\sigma = \frac{\sqrt{2h}}{\varepsilon} \cdot \sqrt{\log \frac{5}{4\delta}}$ suffices.) Then GaussTrie guarantees $(2\varepsilon, 2\delta)$-DP.

The proof can be found in Appendix B.1. At a high level, we invoke the privacy offered by the Gaussian mechanism and apply composition over the h levels of the trie. Neither the adversarial offsets $\eta_{p,\mathcal{A}}$ nor the biases α_i affect privacy because they do not influence sensitivity of the computation or the noise generated by the functionality.

We analyze accuracy when bias $= 1$.[7]

5. Note that just outputting the result has a cost of $\log d$ per heavy hitter, which explains why there is still a dependence on $\log d$ in the computation/communication.

6. We let $N(0, \sigma^2)$ denote the standard normal (i.e., Gaussian) distribution with mean 0 and standard deviation σ. Φ denotes the cumulative distribution function (CDF) of the distribution $N(0, 1)$, and Φ^{-1} is the associated quantile function, i.e., $\Pr_{x\leftarrow N(0,1)}[x \leq \Phi^{-1}(p)] = p$.

7. We perform an empirical evaluation for $\mathsf{bias} = 0$ in Section 6.2. We also note that $\mathsf{bias} = 0$ will result in $t - \gamma$-heavy-hitter detection, for some positive γ.

Ideal Functionality 1: GaussTrie

```
Input: $\bar{X} \in ([b]^h)^n$
Output: $S \subseteq [b]^h$ (heavy-hitters)
1 Parameters: domain $[b]^h$, threshold $t \in \mathbb{N}$, variance
    $\sigma^2 > 0$, target error rate $\beta$, flag bias $\in \{0,1\}$
2 $S := \emptyset$
3 $Q :=$ queue initialized with the empty string
4 $i := -1$ /* Tree level; prefix length */
5 While $|Q| > 0$
6     $q :=$ head of $Q$
7     If $i \neq |q|$ :
          /* Update for new level */
8         $i := |q|$
9         $n_i := |Q|$ (# of prefixes w/ length $i$)
10        $\gamma_i := (2h \cdot b \cdot n_i/\beta)^{-1}$
11        $\alpha_i := \begin{cases} \sqrt{2}\sigma \cdot \Phi^{-1}(\gamma_i) & \text{if bias} = 1 \\ 0 & \text{otherwise} \end{cases}$
12    Dequeue $q$ from $Q$
13    For char $\in [b]$
14        Create $p$ by appending char to $q$
15        $c_p :=$ count of strings in input w/ prefix $p$
16        If $\mathcal{A}$ is passive: $\eta_{p,\mathcal{A}} \leftarrow N(0,\sigma^2)$
17        Else
18            ▶ $\mathcal{A}$ sends $\eta_{p,\mathcal{A}} \in \mathbb{Z}$ or ABORT
19            If ABORT was sent, halt
20        $\eta_p \leftarrow N(0,\sigma^2)$
21        $\tilde{c}_p := c_p + \eta_p + \eta_{p,\mathcal{A}} + \alpha_i$
22        ▶ Send $p, \tilde{c}_p$ to $\mathcal{A}$
23        If $\tilde{c}_p \geq t$ :
              /* At leaves? Add to output */
24            If $|p| = h$: Add $p$ to $S$
              /* Otherwise enqueue */
25            Else: Enqueue $p$ in $Q$
26 Output $S$ (and ▶ send $S$ to $\mathcal{A}$)
```

Theorem 7 (Accuracy of GaussTrie). *GaussTrie with* bias *set to 1 performs t-heavy-hitter detection with error margin*

$$4\sigma \cdot \sqrt{\ln\left(\sqrt{\frac{2}{\pi}} \cdot \frac{\mathsf{Heavy}_t(\bar{X}) \cdot b \cdot h}{\beta}\right)}$$

and error rate β. Moreover, the probability that GaussTrie enqueues a prefix having frequency $< t$ is $\leq \beta$.

We defer the proof to Appendix B.1. At a high level, we use concentration bounds to argue that sufficiently frequent strings are detected and infrequent ones are not.

4.2. Pruning and Inverting Hashes

Here we describe and analyze PrunedInvert, a DP algorithm for finding a preimage of a hashed value s in the input $\bar{X}$. We assume d is a power of 2 for simplicity, so inputs in the dataset can be represented as $\log_2 d$-bit binary strings. We write $x[j]$ for the jth bit of binary string x.

PrunedInvert uses the truncated discrete Laplace distribution.

Definition 8. For $r \in (0,1)$ and $B \in \mathbb{N}$, the *discrete Laplace distribution with rate r truncated to B* (denoted by $\mathbf{dLap}(r,B)$) has probability mass function $f_{\mathbf{dLap}(r,B)}(i) \propto r^{|i|}$ for $i \in \{-B,\dots,B\}$.

Ideal Functionality 2: PrunedInvert

```
Input: $\bar{X} \in (\{0,1\}^{\log_2 d})^n$; hashed value $s$
Output: Either $\bot$ or $y \in \{0,1\}^{\log_2 d}$
1 Parameters: hash function $H$, $t \in \mathbb{N}$, $\varepsilon > 0$,
    $\delta \in (0,1)$
2 tail $:= \lceil \frac{1}{\varepsilon} \ln \frac{1}{\delta} \rceil$
3 Let $X_s := \{x \in \bar{X} : H(x) = s\}$
4 If $\mathcal{A}$ is passive :
5     offset$[j,v] := 0$ for all $j \in [\log_2 d], v \in \{0,1\}$
6 Else
7     ▶ $\mathcal{A}$ sends $\{\text{offset}[j,v] \in \mathbb{Z}\}_{j,v}$ or ABORT
8     If ABORT was sent, halt
  /* Collect votes for preimage's bits */
9 For $j \in [\log_2 d]$
10    count$[j,0] :=$ offset$[j,0] + |\{x \in X_s : x[j] = 0\}|$
11    count$[j,1] :=$ offset$[j,1] + |\{x \in X_s : x[j] = 1\}|$
12    If count$[j,0] \geq$ count$[j,1]$: $y[j] := 0$
13    Else: $y[j] := 1$
  /* Check if smallest margin of victory
     is too small */
14 gap $:= \min_j |\text{count}[j,0] - \text{count}[j,1]|$
15 $\eta \leftarrow \mathbf{dLap}(\exp(-\varepsilon), \text{tail})$
16 If gap $+ \eta \leq t +$ tail, then prune $:= 1$ else
    prune $:= 0$
  /* If smallest margin of victory is too
     small, inverse may violate privacy */
17 If prune $= 1$: $z := \bot$
18 Else
      /* Here, inversion is stable */
19    $z := y$
20 Output $z$ (and ▶ send $z$ to $\mathcal{A}$)
```

As discussed in Section 3.2.3, our functionality finds a preimage y of s bit-by-bit. Specifically, let X_s be the multiset of elements of $\bar{X}$ that are preimages of s. Then for $j = 1,\dots,\log_2 d$ the functionality sets the jth bit of y equal to the "majority vote" for the jth bit among elements of X_s, breaking ties arbitrarily. The adversary is allowed to bias these votes. Nevertheless, a single-client change can only alter the returned preimage if any of the votes are close, i.e., if gap is small. To achieve differential privacy, we cannot prune based on a strict threshold for gap; thus, the functionality computes a noisy estimate of gap and bases the decision to prune on that estimate.

Privacy. Our privacy analysis makes no assumptions on H and, in particular, holds even if it is easy to find collisions in H. Because we will do a careful privacy accounting when we use PrunedInvert as a subroutine in our larger framework, we prove separate DP bounds corresponding to different possibilities for s, based on the following definition:

Definition 9. Fix $X \sim X'$ with $X \Delta X' = \{x, x'\}$ such that $x \in X$ and $x' \in X'$. We say that s

- *does not match the targets* if $s \neq H(x)$ and $s \neq H(x')$
- *matches one target* if $s = H(x)$ but $s \neq H(x')$, or vice versa
- *matches both targets* if $s = H(x)$ and $s = H(x')$.

Fix some $X \sim X'$ as above. We write $z, \mathsf{gap}, \mathsf{prune}$ (resp., $z', \mathsf{gap}', \mathsf{prune}'$) to denote the values of internal variables when the honest input is X (resp., X'). For each of the above possibilities for s, we quantify closeness of the distributions of z, z'.

Claim 10. *If s does not match the targets, then z and z' are identically distributed.*

Proof. Observe that all entries of the array count are the same regardless of whether the honest inputs are X or X'. All other variables are derived by post-processing count. □

Claim 11. *If s matches one target, then $z \approx_{\varepsilon,\delta} z'$.*

A full proof appears in Appendix B.2, but we sketch the main ideas here.

Proof Sketch. Note that $|\mathsf{gap} - \mathsf{gap}'| \leq 1$. This means that $\mathsf{gap} + \eta \approx_{\varepsilon,\delta} \mathsf{gap}' + \eta$ when $\eta \leftarrow \mathbf{dLap}(-\exp(\varepsilon), \mathsf{tail})$. As a result, $\mathsf{prune} \approx_{\varepsilon,\delta} \mathsf{prune}'$. Conditioned on $\mathsf{prune} = \mathsf{prune}' = 1$, the distributions of z and z' are identical (since $z = z' = \perp$). Although the distributions of z and z' may not be close if $\mathsf{prune} = \mathsf{prune}' = 0$, those events each only occur with probability at most δ. □

Claim 12. *If s matches both targets, then $z \approx_{2\varepsilon,2\delta} z'$.*

Proof. Consider a dataset X'' that differs from X, X' in a single element $x'' \notin \{x, x'\}$ with $s \neq H(x'')$, and let z'' be the corresponding random variable. Claim 11 shows that $z \approx_{\varepsilon,\delta} z''$ and $z' \approx_{\varepsilon,\delta} z''$. Combining these two yields the desired result. □

Correctness. We next show correctness of PrunedInvert when there is a unique preimage of s in $\bar{X}$.

Theorem 13 (Correctness of PrunedInvert)**.** *Assume there is a unique $x \in \bar{X}$ with $H(x) = s$, and that x appears at least $t + 2 \cdot \mathsf{tail}$ times in $\bar{X}$. If $\mathcal{A}$ is passive,* PrunedInvert$(\bar{X}, s)$ *outputs $z := x$ with probability 1.*

Proof. Because all offset values are 0 and x is the unique preimage of s in $\bar{X}$, for all $j \in [\log_2 d]$ the value of $\mathsf{count}[j, x[j]]$ is the number of times x appears in $\bar{X}$, and $\mathsf{count}[j, 1 - x[j]] = 0$. So $\mathsf{gap} > 2 \cdot \mathsf{tail}$. Because $\eta \in [-\mathsf{tail}, +\mathsf{tail}]$, line 16 ensures $\mathsf{prune} = 0$. Hence, the output is $z = x$. □

4.3. Hash-Prune-Invert

Here we fully describe the hash-prune-invert (HPI) functionality, which relies on GaussTrie and PrunedInvert using a random hash function. HPI is generic, and supports any heavy-hitter detection algorithm, but we focus on its instantiation with GaussTrie for concreteness.

Ideal Functionality 3: HPI+GaussTrie

```
   Input: $\bar{X} \in ([b]^h)^n$
   Output: $S \subseteq [d]$
1  Parameters: $b, h, k, t \in \mathbb{N}$, $\varepsilon, \sigma^2 > 0$, $\beta, \delta \in (0, 1)$,
     bias $\in \{0, 1\}$
2  Sample $H$ from a family $\mathcal{H} = \{H : [b]^h \to [b]^k\}$
3  ▶ Send $H$ to $\mathcal{A}$
4  If $\mathcal{A}$ is not passive :
5      ▶ $\mathcal{A}$ sends ABORT or CONTINUE
6      If ABORT was sent: halt
7  Compute $S_{\text{init}} := \mathsf{GaussTrie}(H(\bar{X}))$ using
     parameters $b, k, t, \sigma^2, \beta, \mathsf{bias}$
8  $S_{\text{final}} := \emptyset$
9  For $s \in S_{\text{init}}$
10     Compute $z := \mathsf{PrunedInvert}(\bar{X}, s)$ using
         parameters $H, t, \varepsilon, \delta$
11     If $z \neq \perp$ and $H(z) = s$, add $z$ to $S_{\text{final}}$
       /* (Optional:) If $z \neq \perp$ but $H(z) \neq s$,
          log error msg. */
12 Output $S_{\text{final}}$ (and ▶ send $S_{\text{final}}$ to $\mathcal{A}$)
```

Theorem 14 (Privacy of HPI+GaussTrie)**.** *If σ is set as in Theorem 6, then HPI+GaussTrie satisfies $(4\varepsilon, 4\delta)$-DP.*

Proof. Theorem 6 shows that the invocation of GaussTrie satisfies $(2\varepsilon, 2\delta)$-DP. We show that for any H, S_{init} the computation (in the for-loop) of S_{final} satisfies $(2\varepsilon, 2\delta)$-DP. Basic composition concludes the proof.

For $X \sim X'$ and any H, S_{init}, there are three cases: (1) every $s \in S_{\text{init}}$ does not match the targets, (2) at most two values $s_1, s_2 \in S_{\text{init}}$ each match one target, or (3) exactly one $s \in S_{\text{init}}$ matches both targets (cf. Def. 9). Claim 10 implies there is no leakage in the first case. In the second case, Claim 11 implies that each invocation of PrunedInvert on s_1, s_2 satisfies (ε, δ)-DP; basic composition completes the claim. In the third case, the invocation of PrunedInvert ensures $(2\varepsilon, 2\delta)$-DP via Claim 12. $\square$

We analyze correctness of when $\mathcal{A}$ is passive, as an adversary who actively corrupts a server can simply abort the computation. Even then, HPI+GaussTrie only guarantees correctness when there are no hash collisions in the input. To ensure that (with high probability) there are no collisions, we sample H from a family of universal hash functions and restrict attention to the case where no clients are corrupted (so the adversary cannot choose the inputs of corrupted clients after seeing H). In Appendix C, we explore defenses against corrupted clients.

Definition 15. $\mathcal{H} = \{H : [b]^h \to [b]^k\}$ is a *universal* hash family if, for any distinct $x_1, x_2 \in [b]^h$,

$$\Pr_{H \leftarrow \mathcal{H}}[H(x_1) = H(x_2)] = 1/b^k.$$

Theorem 16 (Accuracy of HPI)**.** *Let $\mathcal{H}$ be a universal hash family with $k \geq \log_b(n^2/2\gamma)$, and set* $\mathsf{bias} = 1$. *If no clients are corrupted, HPI performs t-heavy-hitter detection with error margin*

$$\Delta = \max \left\{ 2 \cdot \left\lceil \frac{1}{\varepsilon} \ln \frac{1}{\delta} \right\rceil, \; 4\sigma \cdot \sqrt{\ln \left(\sqrt{\frac{2}{\pi}} \frac{\mathsf{Heavy}_t(\bar{X}) \cdot b \cdot k}{\beta} \right)} \right\}$$

and error rate $\leq \beta + \gamma$.

The proof can be found in Appendix B.3.

5. A Two-Server Protocol

In the previous section we described ideal functionalities for the HPI framework and the underlying heavy-hitter detection algorithm GaussTrie. In this section we discuss how HPI+GaussTrie can be implemented efficiently by a two-server protocol.

The protocol is given in Figure 1, and relies on the subroutines in Figures 2 and 3. We give an outline of the overall protocol here, and describe the components of the protocol in more detail in the subsections that follow:

1) The servers jointly sample $H : [b]^h \to [b]^k$, and send it to the clients.[8]
2) A client with input $x_i \in [b]^h$ computes $h_i := H(x_i) \in [b]^k$. It then generates incremental DPF keys $\mathsf{key}_{i,0}, \mathsf{key}_{i,1}$ satisfying Eq. (1), and sends $\mathsf{key}_{i,j}$ to server j. We define $\mathsf{Eval}_j(p) = \sum_{i \in [n]} \mathsf{Eval}_{\mathsf{key}_{i,j}}(p)$.
3) The servers run a 2PC protocol to evaluate GaussTrie (cf. Figure 2 and Section 5.1) and obtain output S_{init}.
4) For each element of S_{init}, the servers run a 2PC protocol to evaluate PrunedInvert (cf. Figure 3 and Section 5.2) and obtain output S.

We defer a discussion of the security of the protocol to Section 5.4.

```
Overall protocol

Parties: Two non-colluding servers; $n$ clients
Input: Client $i$ holds $x_i$
1) Using a coin-tossing protocol the servers select a hash function $H$ and
   send it to each client
2) Each client $i$ does:
   a) Set $h_i := H(x_i)$
   b) Compute DPF keys $\mathsf{key}_{i,0}, \mathsf{key}_{i,1}$ as described in the text, and send
      $\mathsf{key}_{i,j}$ to server $j \in \{0, 1\}$
3) The servers then do:
   a) Run protocol GaussTrie to obtain $S_{\text{init}}$
   b) Set $S := \emptyset$. Then for $s \in S_{\text{init}}$ do:
      i) Run protocol PrunedInvert on input $s$ to obtain output $z$
      ii) If $z \neq \perp$ and $H(z) = s$, add $z$ to $S$. (If $z \neq \perp$ but $H(z) \neq s$
          an optional error message can be output.)
   c) Output $S$
```

Figure 1: Two-server protocol computing HPI+GaussTrie.

```
GaussTrie

Parties: Two non-colluding servers
Parameters: $b, k, t, \sigma^2, \beta$, bias
Input: Server $j$ holds $\{\mathsf{key}_{i,j}\}_{i \in [n]}$
1) Each server initializes $S := \emptyset$, $i := -1$, and $Q$ to a queue initialized
   with the empty string.
2) While $|Q| > 0$, the servers then do:
   a) Set $q :=$ head of $Q$
   b) If $i \neq |q|$, set:
      i) $i := |q|$
      ii) $n_i := |Q|$
      iii) $\gamma_i := (2h \cdot b \cdot n_i / \beta)^{-1}$
      iv) $\alpha_i := \begin{cases} \sqrt{2}\sigma \cdot \Phi^{-1}(\gamma_i) & \text{if bias} = 1 \\ 0 & \text{otherwise} \end{cases}$
   c) Dequeue $q$ from $Q$
   d) For char $\in [b]$
      i) Create $p$ by appending char to $q$
      ii) Server $j$ sets $c_{p,j} := \mathsf{Eval}_j(p)$
      iii) Sample $\eta_{p,j} \leftarrow N(0, \sigma^2)$ and set $\tilde{c}_{p,j} := c_{p,j} + \eta_{p,j}$
      iv) Send $\tilde{c}_{p,j}$ to the other server; set $\tilde{c}_p := \tilde{c}_{p,0} + \tilde{c}_{p,1} + \alpha_i$
      v) If $\tilde{c}_p \geq t$ and $|p| = h$: Add $p$ to $S$
      vi) If $\tilde{c}_p \geq t$ and $|p| < h$: Enqueue $p$ in $Q$
3) Output $S$
```

Figure 2: Protocol computing GaussTrie.

8. To prevent inconsistencies, each server can sign H before sending it to the clients; if any client receives inconsistent (signed) H's from the servers, it echos them to the servers who then abort. For simplicity we omit this from the protocol description.

PrunedInvert

Parties: Two non-colluding servers

Input: Server j holds $\{\mathsf{key}_{i,j}\}_{i\in[n]}$; both servers hold a hash value s.

1) Server j sets $c_j := \mathsf{Eval}_j(s) \in \{0,1\}^{1+2\log d}$
2) Using an actively secure, generic 2PC protocol, the servers evaluate the randomized functionality $f(c_0, c_1)$ defined as:
   a) Set $c := c_0 + c_1$
   b) For $0 < k \leq \log_2 d$, do:
   $$y[k] := \begin{cases} 0 & c[2k] \geq c[2k+1] \\ 1 & \text{otherwise} \end{cases}$$
   c) $\mathsf{gap} := \min_k |c[2k+1] - c[2k]|$
   d) $\eta \leftarrow \mathbf{dLap}(\exp(-\epsilon), \mathsf{tail})$
   e) If $\mathsf{gap} + \eta \leq t + \mathsf{tail}$, return $\perp$; else return y

Figure 3: Protocol computing PrunedInvert.

5.1. Securely Computing GaussTrie

Securely computing the GaussTrie algorithm is fairly straightforward given the incremental DPFs set up by the users. When processing prefix p in an execution of the inner loop of GaussTrie (cf. Ideal Functionality 1), each server $j \in \{0,1\}$ (independently) samples $\eta_{p,j}$ and locally computes[9] and releases $\mathsf{Eval}_j(p) + \eta_{p,j}$. The servers sum these values to obtain $c_p + \eta_{p,0} + \eta_{p,1} + \alpha_i$, and compare the result to the threshold t. Note that since all decisions about placing elements in Q or S are known to both servers, they can each locally compute all the necessary values.

We stress that it is not necessary to generate the noise $\eta_{p,0}, \eta_{p,1}$ securely since Theorem 6 shows that privacy holds as long as either $\eta_{p,0}$ or $\eta_{p,1}$ is generated correctly, which is true whenever at least one server is honest.

5.2. Securely Computing PrunedInvert

In contrast to GaussTrie, the noise for computing PrunedInvert (i.e., η in Figure 3) must be generated securely. If an adversary could make η large (even if unpredictable) then it could force a non-heavy hitter $y \neq \perp$ to be returned. If no honest party has a value that hashes to s, then $y = 0^{\log d}$, but if one party has such a value then y will take that value. This distinguishes two neighboring datasets with certainty, violating differential privacy.

The protocol computing PrunedInvert proceeds as follows. Each server uses $\mathsf{Eval}_j(s)$ as an additive share of a vector c encoding "votes" for each bit of a preimage of s (cf. Eq. (1)). The servers then securely compute a (randomized) function f that does the following:

1) Reconstruct c and compute, for each position in the binary expansion of a possible preimage y, whether 0 or 1 is more common.
2) Compute the smallest margin gap and sample η.
3) If $\mathsf{gap} + \eta < t + \mathsf{tail}$, prune s (i.e., output nothing). Otherwise output y.

We defer to Section 5.3 a discussion of how to construct a circuit for sampling η; note that the rest of f involves only simple computations such as additions and comparisons.

9. When $|p| = k$, the servers use only the first field of $\mathsf{Eval}(p)$.

5.3. Secure Noise Generation

We consider two approaches for securely sampling from $\mathbf{dLap}(r, \mathsf{tail})$, assuming the availability of uniform bits (which are easy to generate using 2PC). Note that since $\Pr_{\eta\leftarrow\mathbf{dLap}(r)}[|\eta| > \mathsf{tail}] = O(\delta)$, it suffices to sample from $\mathbf{dLap}(r)$.

Inversion sampling. The first approach is to use inversion sampling. Let $F_{\mathbf{dLap}(r)}$ be the CDF of $\mathbf{dLap}(r)$. Then we sample uniform $u \in [0,1]$ (of some fixed precision ℓ; see below), and let $\eta \in \mathbb{Z}$ be the smallest value for which $F_{\mathbf{dLap}(r)}(\eta) \geq u$. To implement the latter securely, we can let F be a public array containing the values of $F_{\mathbf{dLap}(r)}(i)$ for $-\mathsf{tail} \leq i \leq \mathsf{tail}$, and then do a linear scan to find the least index η with $F(\eta) \geq u$. As an optimization, we can exploit the fact that $\mathbf{dLap}$ is symmetric; thus, it suffices to use the array $F_{|\mathbf{dLap}(r)|}(i)$ for $0 \leq i \leq \mathsf{tail}$, and then use one additional bit to determine the sign of η.

We set $\ell = O(\log(\mathsf{tail}/\delta))$. This ensures that the total variation distance between $\mathbf{dLap}$(r) and the actual distribution we sample from is bounded by $O(\delta)$, and suffices for $(\varepsilon, O(\delta))$-DP since in the analysis of differential privacy we only care about the effect of the noise on the (hashed) input of one client.

Direct sampling. Dwork et al. [27], observed that sampling from $\mathbf{dLap}(r)$ reduces to sampling once from the geometric distribution $\mathbf{Geo}(1-r)$, once from a biased Bernoulli distribution, and once from the unbiased Bernoulli distribution. In turn, sampling from $\mathbf{Geo}(1-r)$ with ℓ-bit precision can be reduced to sampling from ℓ biased Bernoulli distributions. Sampling from a biased Bernoulli distribution with ℓ-bit precision can be done by sampling a uniform value in $[0,1]$ with $\ell' = \lceil -\log_2(\delta/(\ell+1)) \rceil$ bits of precision and comparing it to the desired bias. This incurs $< \delta/\ell$ error for each sample and thus $O(\delta)$ error overall for $O(\ell)$ samples.

For practical settings of the parameters, we find that securely sampling $\mathbf{dLap}(r)$ using direct sampling is substantially more efficient than using inversion sampling.

5.4. Security Guarantees

Accuracy of the protocol when no clients are corrupted and the adversary is passive (and a universal hash family is used) follows from Theorem 16. In Appendix C we discuss the effect of malicious clients.

ε	Circuit size (non-XOR gates)	Running time (ms)
1	45573	153
2	45797	147
4	45495	136
8	45463	136

TABLE 2: Running times and circuit sizes for different values of ε, for a single run of PrunedInvert. We allow for 1M clients , each holding a 256-bit string (i.e., $\log_2 d = 256$).

We next argue that our protocol achieves computational differential privacy against an active adversary corrupting one of the servers and any number of clients. Theorem 14 already shows that a centralized version of the HPI+GaussTrie algorithm satisfies $(4\epsilon, 4\delta)$-DP. Our protocol securely computes that algorithm, with the only discrepancy being that samples of η are within $O(\delta)$ statistical difference of the correct distribution. We stress in particular that this holds even though malicious clients may generate incorrect DPF keys, since our ideal functionalities for GaussTrie and PrunedInvert already allow the adversary to choose arbitrary offsets for all values computed using the DPF keys in the protocol.

Theorem 17. *The protocol in Figure 1, implemented as described above and with parameters matching those of Theorem 14, satisfies $(4\epsilon, O(\delta))$-DP against an active adversary corrupting one server and an arbitrary number of clients.*

6. Experimental Results

6.1. Implementation

The main overhead of our solution compared to Poplar (using domain $[b]^k$) is from the 2PC protocol for securely computing PrunedInvert, and so we focus on that aspect of our protocol. We implement the required 2PC protocol for PrunedInvert in C++ using the state-of-the-art authenticated garbling scheme [28] in the EMP toolkit [29]. The circuit itself was generated using CBMC-GC [30], [31].[10]

In Table 2 we show experimental results for a single run of PrunedInvert, with both servers on the same machine, each using a single core. Since we use garbled circuits, our protocol has $O(1)$ round complexity, so latency is not expected to be an issue in practice. Overall, our results indicate that our protocol adds little overhead as compared to Poplar. For comparison, the implementation of Poplar achieves a throughput of 124 clients per second for a 256-bit domain [4]. Note further than PrunedInvert only needs to be computed once per heavy hitter.

10. Available at https://gitlab.com/securityengineering/CBMC-GC-2.

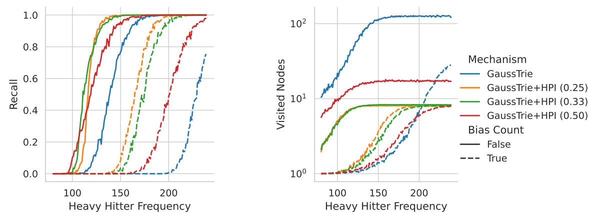

Figure 4: Identifying a single heavy hitter.

6.2. Utility

We compare the utility of GaussTrie and HPI+GaussTrie using synthetic data. In the first experiment we consider a dataset with a single heavy hitter of varying frequency, and evaluate the average recall (i.e., the probability of correctly identifying the heavy hitter). In the second experiment we consider a dataset where element frequencies follow a power-law distribution, and evaluate the F1 score as a function of the heavy-hitter threshold. In both cases we evaluate HPI+GaussTrie with and without biasing the noisy counts; this leads to different trade-offs between utility and computation, where the latter is measured by the total number of trie nodes expanded. We also explore the effect of different splits between the privacy budgets of HPI and GaussTrie, with HPI being allocated $1/2$, $1/3$, or $1/4$ of the total privacy budget.

In all our experiments, we set $k = \log_b(n^2) + 40/\log_2(b)$ to guarantee that the probability of a collision is smaller than $\min\{2^{-40}, 1/n\}$. All results are averaged over 500 independent runs and satisfy $\delta \leq (1+e)/n$ and failure rate $\beta \leq 1/10$.

Single heavy hitter. We create datasets of $n = 10^3$ strings, each 256 bits long, where each dataset contains a single heavy hitter whose frequency varies from 80–240. We fix the threshold $t = 80$, and set $\varepsilon = 2$. We set $b = 256$ and $h = 32$, so $k = 8$. We evaluate the average recall (i.e., probability of correctly identifying the heavy hitter) and number of expanded trie nodes. Results are presented in Figure 4. We observe that:

- Whether or not we bias counts, HPI+GaussTrie has superior recall compared to GaussTrie. This is due to the reduction in the error margin (cf. Remark 5).
- Biasing the counts degrades recall, but also improves efficiency by reducing the number of visited nodes.
- When not biasing the counts, HPI+GaussTrie is superior to GaussTrie in terms of *both* recall and efficiency.
- As we reduce the privacy budget allocated to HPI, recall improves while efficiency gets worse. This effect is more pronounced when bias is added.

Power-law dataset. We generate datasets of $n = 10^5$ items, each 256 bits long, where items follow a power-law (Pareto)

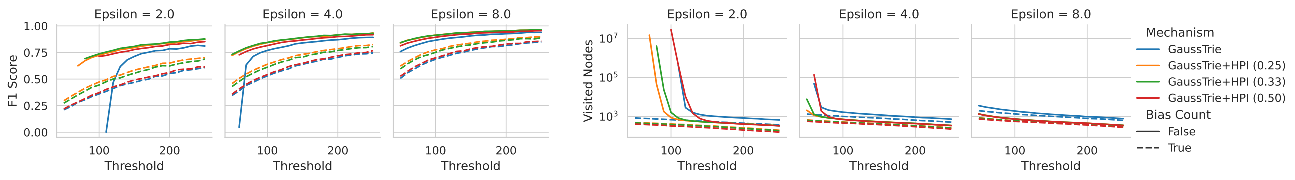

Figure 5: Identifying heavy hitters in a power-law dataset.

distribution with parameter 1.25. This ensures a small number of elements have high frequency while the vast majority have low frequency, and is representative of real datasets of interest for heavy-hitter identification. We again set $b = 256$ and $h = 32$; now $k = 10$. We vary the threshold t from 50–250, and run the experiment with $\varepsilon \in \{2, 4, 8\}$. Results are presented in Figure 5. Overall, we observe that (when not biasing the counts) HPI+GaussTrie performs better than GaussTrie in terms of both accuracy and efficiency across all values of ϵ, with the benefit being more pronounced in the high-privacy regime. We also find that biasing the counts can significantly improve efficiency when the threshold is low (so the number of heavy hitters is large).

References

[1] D. Mouris, C. Patton, H. Davis, P. Sarkar, and N. G. Tsoutsos, "Mastic: Private weighted heavy-hitters and attribute-based metrics," *Proc. Priv. Enhancing Technol.*, 2025.

[2] "Network Error Logging," https://www.w3.org/TR/network-error-logging, W3C working draft, accessed 11-14-2024.

[3] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI*, 2017, pp. 259–282.

[4] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *IEEE Security & Privacy*. IEEE, 2021, pp. 762–776.

[5] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via boolean shares," in *SCN*, 2022.

[6] H. Davis, C. Patton, M. Rosulek, and P. Schoppmann, "Verifiable distributed aggregation functions," *Proc. Priv. Enhancing Technol.*, 2023.

[7] D. Mouris, P. Sarkar, and N. G. Tsoutsos, "Plasma: Private, lightweight aggregated statistics against malicious adversaries," *Proc. Priv. Enhancing Technol.*, 2024.

[8] R. Barnes, D. Cook, C. Patton, and P. Schoppmann, "Verifiable distributed aggregation functions," Internet Engineering Task Force, Internet-Draft draft-irtf-cfrg-vdaf-13, 2024, available at `https://datatracker.ietf.org/doc/draft-irtf-cfrg-vdaf`.

[9] "Divvi up: A privacy respecting telemetry service," 2024, available at `https://datatracker.ietf.org/wg/ppm/about/`.

[10] A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas, "Releasing search queries and clicks privately," in *Proc. 18th Intl. Conf. on the World Wide Web*. ACM, 2009, p. 171–180.

[11] M. Bun, K. Nissim, and U. Stemmer, "Simultaneous private learning of multiple concepts," in *Innovations in Theoretical Computer Science*. ACM, 2016.

[12] S. Vadhan, "The complexity of differential privacy," *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pp. 347–450, 2017.

[13] R. Bassily and A. Smith, "Local, private, efficient protocols for succinct histograms," in *STOC*, 2015, pp. 127–135.

[14] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta, "Practical locally private heavy hitters," *J. Machine Learning Research*, vol. 21, no. 16, pp. 1–42, 2020.

[15] V. Balcer and A. Cheu, "Separating local & shuffled differential privacy via histograms," in *1st Conference on Information-Theoretic Cryptography (ITC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[16] J. Bell, A. Gascón, B. Ghazi, R. Kumar, P. Manurangsi, M. Raykova, and P. Schoppmann, "Distributed, private, sparse histograms in the two-server model," in *ACM Conf. on Computer and Communications Security*. ACM, 2022, pp. 307–321.

[17] L. Braun, A. Gascón, M. Raykova, P. Schoppmann, and K. Seth, "Malicious security for sparse private histograms," 2024, available at https://eprint.iacr.org/2024/469.

[18] F. B. Durak, C. Weng, E. Anderson, K. Laine, and M. Chase, "Precio: Private aggregate measurement via oblivious shuffling," *Cryptology ePrint Archive*, 2021.

[19] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *Crypto*, 2009, pp. 126–142.

[20] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Adv. in Cryptology—Eurocrypt 2014*, ser. LNCS. Springer, 2014, pp. 640–658.

[21] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Adv. in Cryptology—Eurocrypt 2015*, ser. LNCS. Springer, 2015, pp. 337–367.

[22] ——, "Function secret sharing: Improvements and extensions," in *ACM Conference on Computer and Communications Security (CCS)*, 2016, pp. 1292–1303.

[23] J. Zhang, X. Xiao, and X. Xie, "PrivTree: A differentially private algorithm for hierarchical decompositions," in *SIGMOD*. ACM, 2016, pp. 155–170.

[24] W. Zhu, P. Kairouz, B. McMahan, H. Sun, and W. Li, "Federated heavy hitters discovery with differential privacy," in *23rd Intl. Conf. on Artificial Intelligence and Statistics (AISTATS)*, ser. Proc. Machine Learning Research, vol. 108. PMLR, 2020, pp. 3837–3847, available at `https://arxiv.org/abs/1902.08534`.

[25] G. Cormode and A. Bharadwaj, "Sample-and-threshold differential privacy: Histograms and applications," in *Intl. Conf. on Artificial Intelligence and Statistics (AISTATS)*, ser. Proc. Machine Learning Research, vol. 151. PMLR, 2022, pp. 1420–1431, available at `https://arxiv.org/abs/2112.05693`.

[26] K. Chadha, J. Chen, J. Duchi, V. Feldman, H. Hashemi, O. Javidbakht, A. McMillan, and K. Talwar, "Differentially private heavy hitter detection using federated analytics," in *IEEE Conf. on Secure and Trustworthy Machine Learning (SaTML)*. IEEE, 2024, pp. 512–533, available at `https://arxiv.org/abs/2307.11749`.

[27] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Advances in Cryptology—Eurocrypt 2006*. Springer, 2006, pp. 486–503.

[28] X. Wang, S. Ranellucci, and J. Katz, "Authenticated garbling and efficient maliciously secure two-party computation," in *CCS*. ACM, 2017, pp. 21–37.

[29] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient multi-party computation toolkit," https://github.com/emp-toolkit.

[30] N. Büscher, A. Holzer, A. Weber, and S. Katzenbeisser, "Compiling low depth circuits for practical secure computation," in *ESORICS, Part II*, ser. LNCS, vol. 9879. Springer, 2016, pp. 80–98.

[31] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith, "Secure two-party computations in ANSI C," in *CCS*. ACM, 2012, pp. 772–783.

[32] J. Dong, A. Roth, and W. J. Su, "Gaussian differential privacy," *CoRR*, vol. abs/1905.02383, 2019.

[33] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[34] L. Wasserman, "Stat 700 lecture notes 5," available at `https://www.stat.cmu.edu/~larry/=stat700/Lecture5.pdf`.

Appendix A. Histogram Estimation

We show that a DP heavy-hitter detector implies a DP histogram estimator.

Definition 18. An algorithm performs *histogram estimation with ℓ_∞ error Δ and error rate β* if, on any multiset X, it produces an associative data structure $a(X)$ such that $\Pr\left[\exists s \in \mathcal{D}\ |c_s(X) - a_s(X)| > \Delta\right] < \beta$.

Lemma 19. *If there is an (ε, δ)-DP algorithm that performs heavy-hitter detection with Δ error margin and error rate β, there is a $(2\varepsilon, \delta)$-DP algorithm that performs histogram estimation with ℓ_∞ error $\Delta + \frac{1}{\varepsilon}\ln\frac{n}{\beta}$ and error rate 2β.*

Proof. Let A be the heavy-hitter detection algorithm. Histogram estimator A' does the following: run $A(X)$ with threshold $t = 1$ to create a set S. Then, for $s \in S$, let $a_s = c_s + \mathbf{Lap}(1/\varepsilon)$ where c_s is its true count. For $s \notin S$, set $a_s := 0$.

By definition of heavy-hitter detection, every item that occurs at least $t + \Delta$ times will be in S. Hence, every $s \notin S$ satisfies $c_s \leq \Delta$. Since $a_s = 0$, we have $|c_s - a_s| = |c_s| \leq \Delta$.

By a tail bound on the Laplace distribution and a union bound over the set $\{a_s\}_{s\in S}$, we can conclude that $\Pr\left[\exists s \in S : |c_s - a_s| > \frac{1}{\varepsilon}\ln\frac{|S|}{\beta}\right] < \beta$. After we factor in the β error rate of A, the following holds except with probability $\leq 2\beta$:

$$|c_s - a_s| < \frac{1}{\varepsilon}\ln\frac{|S|}{\beta} + \Delta \leq \frac{1}{\varepsilon}\ln\frac{n}{\beta} + \Delta$$

for all $s \in \mathcal{D}$. □

A corollary of the above is that lower bounds for DP histogram estimation imply lower bounds for DP heavy-hitter detection. We use one by Vadhan [12].

Theorem 20 (Lower bound for DP histograms)**.** *If an (ε, δ)-DP algorithm performs histogram estimation with ℓ_∞ error Δ and error rate $1/10$, then $\Delta \geq \frac{\kappa}{\varepsilon}\min\left(\log\frac{1}{\delta},\ \log d\right)$ for some constant κ.*

Theorem 21 (Lower bound for DP heavy-hitters)**.** *If an $(\varepsilon, \delta = o(1/n))$-DP algorithm solves heavy-hitter detection with error margin Δ and error rate $1/20$, then*

$$\Delta \geq \frac{\kappa}{2\varepsilon}\min\left(\log\frac{1}{\delta},\ \log d\right) - \frac{1}{\varepsilon}\log 20n.$$

Proof. Assume some algorithm A achieves $\Delta^* = \frac{\kappa}{2\varepsilon}\min\left(\log\frac{1}{\delta},\ \log d\right) - \frac{1}{\varepsilon}\log 20n$ error margin and error rate $1/20$. By Lemma 19, there is some $(2\varepsilon, \delta)$-DP histogram estimator with error $\frac{\kappa}{2\varepsilon}\min\left(\log\frac{1}{\delta},\ \log d\right)$. This contradicts Theorem 20. □

Appendix B. Deferred Proofs

B.1. GaussTrie

We first prove that GaussTrie ensures differential privacy, then prove accuracy and efficiency.

Theorem 6 (Privacy of GaussTrie)**.** *Let σ satisfy*

$$\Phi\left(\frac{\sqrt{h}}{2\sigma} - \frac{\varepsilon\sigma}{\sqrt{h}}\right) - e^{\varepsilon}\cdot\Phi\left(-\frac{\sqrt{h}}{2\sigma} - \frac{\varepsilon\sigma}{\sqrt{h}}\right) \leq \delta.$$

(For $\varepsilon < 1$, taking $\sigma = \frac{\sqrt{2h}}{\varepsilon}\cdot\sqrt{\log\frac{5}{4\delta}}$ suffices.) Then GaussTrie guarantees $(2\varepsilon, 2\delta)$-DP.

Proof. We prove (ε, δ)-add/remove DP for datasets X, X' differing in the presence of exactly one entry. We then use the fact that (ε, δ)-add/remove DP implies $(2\varepsilon, 2\delta)$-replacement DP (which is the notion used in this work).

Fix X, X' with $X' = X \cup \{x\}$, and arbitrary inputs $\hat{X}$ for the corrupted clients. Let $\bar{X} = X \cup \hat{X}$. We append a "prime" to a variable to indicate the corresponding variable when the honest parties' input is X' as opposed to X.

Fix some p. If p is not a prefix of x, then $\tilde{c}_p, \tilde{c}'_p$ are identically distributed. If p is a prefix of x, then $c_p + \eta_{p,\mathcal{A}} + \alpha_i$ and $c'_p + \eta_{p,\mathcal{A}} + \alpha_i$ differ by one, and hence $\tilde{c}_p, \tilde{c}'_p$ are each the sum of a 1-sensitive function and a random variable from $N(0, \sigma^2)$. It follows that $\tilde{c}_p, \tilde{c}'_p$ satisfy $1/\sigma$-Gaussian differential privacy [32, Theorem 2.7].

Since there are at most h prefixes of x that are encountered during execution of GaussTrie, the algorithm overall ensures $\sqrt{h}/\sigma$-Gaussian differential privacy [32, Corollary 3.3].

To convert to approximate differential privacy, we simply invoke [32, Corollary 2.13]. In the special case where $\varepsilon \le 1$, we rely on [33, Appendix A]. $\square$

Theorem 7 (Accuracy of $\mathsf{GaussTrie}$)**.** *GaussTrie with* bias *set to 1 performs t-heavy-hitter detection with error margin*

$$4\sigma \cdot \sqrt{\ln\left(\sqrt{\frac{2}{\pi}} \cdot \frac{\mathsf{Heavy}_t(\bar{X}) \cdot b \cdot h}{\beta}\right)}$$

and error rate β. Moreover, the probability that GaussTrie enqueues a prefix having frequency $< t$ is $\le \beta$.

Proof. We first show that $2\alpha_i \le \eta_p + \eta_{p,\mathcal{A}} + \alpha_i \le 0$ for all prefixes p, except with probability at most β. The distribution of $\eta_p + \eta_{p,\mathcal{A}}$ is $N(0, 2\sigma^2)$. By definition of α_i and symmetry of the Gaussian distribution, $|\eta_p + \eta_{p,\mathcal{A}}| \le |\alpha_i|$ for some fixed p except with probability $\le 2\gamma_i = 2 \cdot (\beta/2hbn_i)$. There are $b \cdot n_i$ extensions of enqueued prefixes of length i, so a union bound over all such extensions, and then over all h possible prefix lengths, completes the proof of the claim.

The remainder of the proof conditions on $2\alpha_i \le \eta_p + \eta_{p,\mathcal{A}} + \alpha_i \le 0$ for all p. Since p is enqueued in Q or added to S iff $c_p + \eta_p + \eta_{p,\mathcal{A}} + \alpha_i \ge t$, we see that p is not added to Q or S when $c_p < t$, but is added to Q or S when $c_p \ge t + 2 \cdot \max_i |\alpha_i|$. Hence the error margin is $2 \cdot \max_i |\alpha_i| = \sqrt{2}\sigma \max_i |\Phi^{-1}(\gamma_i)|$.

To find an upper bound on $|\Phi^{-1}(\gamma_i)|$, it suffices to find some $\ell > 0$ for which $\Pr\left[N(0,1) \ge \ell\right] \le \gamma_i$. Mill's inequality [34] implies $\ell = \sqrt{2\ln\left(\frac{1}{\sqrt{2\pi}\gamma_i}\right)}$ works. Substitution of γ_i and then $n_i \le \mathsf{Heavy}_t(X)$ completes the proof of the theorem. $\square$

B.2. PrunedInvert

Claim 11. *If s matches one target, then $z \approx_{\varepsilon,\delta} z'$.*

Proof. It is easy to see that $|\mathsf{gap} - \mathsf{gap}'| \le 1$. Assume w.l.o.g. that $\mathsf{gap} \le \mathsf{gap}'$.

<u>Case 1</u> $\mathsf{gap}' \le t$. Then $\mathsf{prune} = 1$ and $\mathsf{prune}' = 1$ with probability 1, so both z, z' are $\perp$.

<u>Case 2</u> $\mathsf{gap} > t$. Since $t \ge 1$, this means that all the elections to determine the preimages y and y' have unambiguous victors (i.e., there are no ties). We claim this implies $y = y'$. Indeed, because s matches one target, for all j exactly one of $\mathsf{count}[j,0], \mathsf{count}[j,1]$ goes up or down by one when switching from $\bar{X}$ to $\bar{X}'$. But the only way this could cause a change in the result would be by breaking or causing a tie, and we have just noted that no ties occur.

Since $y = y'$, the values z, z' are the same when $\mathsf{prune} = \mathsf{prune}' = 0$. Thus, it suffices to argue that $\mathsf{prune} \approx_{\varepsilon,\delta} \mathsf{prune}'$. We do this by showing $\mathsf{gap}+\eta \approx_{\varepsilon,\delta} \mathsf{gap}'+\eta$ and invoking the data-processing inequality. If $\mathsf{gap} = \mathsf{gap}'$ this is immediate, so we assume $\mathsf{gap}' = \mathsf{gap} + 1$.

Observe that, for any v and $\eta \leftarrow \mathbf{dLap}(\exp(-\varepsilon), \mathsf{tail})$,

$$\begin{aligned} \Pr_\eta\left[\mathsf{gap} + \eta = v\right] &= \Pr_\eta\left[\eta = v - \mathsf{gap}\right] \\ \text{and } \Pr_\eta\left[\mathsf{gap}' + \eta = v\right] &= \Pr_\eta\left[\eta = v - \mathsf{gap}'\right] \\ &= \Pr_\eta\left[\eta = v - \mathsf{gap} - 1\right]. \end{aligned}$$

If both $v - \mathsf{gap}$ and $v - \mathsf{gap} - 1$ are outside $[-\mathsf{tail}, \mathsf{tail}]$, both probabilities are 0. If both are in $[-\mathsf{tail}, \mathsf{tail}]$, the definition of $\mathbf{dLap}(\exp(-\varepsilon), \mathsf{tail})$ implies the probabilities differ by a multiplicative factor of $\exp(\varepsilon)$. Otherwise, one probability is 0 while the other is

$$\begin{aligned} \Pr_\eta\left[\eta = \mathsf{tail}\right] &= \frac{\exp(-\mathsf{tail} \cdot \varepsilon)}{\sum_{i=-\mathsf{tail}}^{\mathsf{tail}} \exp(-|i| \cdot \varepsilon)} \\ &\le \exp(-\mathsf{tail} \cdot \varepsilon) \\ &\le \delta, \end{aligned} \tag{2}$$

where the first inequality uses the fact that i can take on value 0 and all terms in the summation are positive. The second inequality comes from our relatively large choice of tail. We conclude that, for any V,

$$\begin{aligned} \Pr_\eta\left[\mathsf{gap} + \eta \in V\right] &= \sum_{v \in V} \Pr_\eta\left[\mathsf{gap} + \eta = v\right] \\ &= \sum_{v \in V} \Pr_\eta\left[\eta = v - \mathsf{gap}\right] \\ &\le \delta + \sum_{v \in V} e^\varepsilon \cdot \Pr_\eta\left[\eta = v - \mathsf{gap} - 1\right] \\ &= e^\varepsilon \cdot \Pr_\eta\left[\mathsf{gap}' + \eta \in V\right] + \delta. \end{aligned}$$

(The additive term δ appears exactly once because there is at most one $v \in V$ for which $\Pr_\eta\left[\mathsf{gap} + \eta = v\right] > 0$ but $\Pr_\eta\left[\mathsf{gap}' + \eta = v\right] = 0$.) A symmetric argument implies $\Pr_\eta\left[\mathsf{gap}' + \eta \in V\right] \le e^\varepsilon \cdot \Pr_\eta\left[\mathsf{gap} + \eta \in V\right] + \delta$.

<u>Case 3</u> $\mathsf{gap}' > t, \mathsf{gap} = t$. Here $z = \perp$, and since $\mathsf{gap}' = t + 1$ we have $z' = \perp$ except with probability at most δ (using (2)). $\square$

B.3. HPI

Theorem 16 (Accuracy of HPI)**.** *Let $\mathcal{H}$ be a universal hash family with $k \ge \log_b(n^2/2\gamma)$, and set* $\mathsf{bias} = 1$*. If no clients are corrupted, HPI performs t-heavy-hitter detection with error margin*

$$\Delta = \max \left\{ 2 \cdot \left\lceil \frac{1}{\varepsilon} \ln \frac{1}{\delta} \right\rceil, \; 4\sigma \cdot \sqrt{\ln\left(\sqrt{\frac{2}{\pi}} \frac{\mathsf{Heavy}_t(\bar{X}) \cdot b \cdot k}{\beta}\right)} \right\}$$

and error rate $\le \beta + \gamma$.

Proof. Because $\mathcal{H}$ is universal, each pair of distinct inputs collides with probability $\leq 2\gamma/n^2$. By a union bound over at most $\binom{n}{2}$ pairs, there are no hash collisions between any pair of inputs except with probability $\leq \gamma$; in what follows we condition on that event.

Theorem 7 shows that, except with probability $\leq \beta$, the set S_{init} contains all hash values that appear at least $t + 4\sigma \cdot \sqrt{\ln\left(\sqrt{\frac{2}{\pi}} \frac{\mathsf{Heavy}_t(\bar{X}) \cdot b \cdot k}{\beta}\right)}$ times in $H(\bar{X})$, and will not contain any hashes appearing $< t$ times. $\mathsf{PrunedInvert}$ is called on each $s \in S_{\text{init}}$. Let x be the (unique) preimage of s. Theorem 13 implies that if x occurs more than $t+2\cdot\lceil \frac{1}{\varepsilon} \ln \frac{1}{\delta} \rceil$ times in $\bar{X}$, it will be output by $\mathsf{PrunedInvert}$ and hence included in S_{final}. $\square$

Appendix C. Correctness of HPI with Corrupted Clients

We consider correctness in the presence of actively corrupted clients (assuming a passively corrupted server). There are two aspects of correctness to consider. The first is *suppression* of true heavy hitters (among the honest clients); the second is *undue influence* in causing items to be identified as heavy hitters (beyond what the adversary can do by simply changing the inputs of malicious clients).

HPI does not prevent suppression of heavy hitters by malicious clients. This is because malicious clients can potentially cause all hashes in S_{init} to be pruned, so none of them will be inverted.

HPI+GaussTrie can be adapted to prevent undue influence. To do so, there are two issues that need to be addressed. First, we need to ensure that malicious clients cannot choose inputs that would cause a collision in H. (Note this is a concern even if clients run the protocol honestly, but may choose their inputs freely during the protocol execution.) This can be done simply by choosing H to be a second-preimage resistant hash function. Second, we need to make sure that each client can contribute at most one (hashed) input. To do so, we can rely on a protocol by Boneh et al. [4] that allows the servers to check that at most one entry of a shared vector is 1 and the rest are 0. The servers can run this protocol level-by-level using the incremental DPF keys provided by each client, rejecting a client's contribution if verification fails at any level.