How to Make Private Distributed Cardinality Estimation Practical, and Get Differential Privacy for Free

Changhui Hu[1], Jin Li[2], Zheli Liu[3,†], Xiaojie Guo[3], Yu Wei[3], Xuan Guang[4], Grigorios Loukides[5]
Changyu Dong[1,†]

[1] *School of Computing, Newcastle University, {changhui.hu,changyu.dong}@newcastle.ac.uk*
[2] *Institute of AI and Blockchain, Guangzhou University, lijin@gzhu.edu.cn*
[3] *College of Cyber Science, Nankai University, liuzheli@nankai.edu.cn*
{xiaojie.guo,stoneboat}@mail.nankai.edu.cn
[4] *School of Mathematical Science, Nankai University, xguang@nankai.edu.cn*
[5] *Department of Informatics, King's College London, grigorios.loukides@kcl.ac.uk*

Abstract

Secure computation is a promising privacy enhancing technology, but it is often not scalable enough for data intensive applications. On the other hand, the use of sketches has gained popularity in data mining, because sketches often give rise to highly efficient and scalable sub-linear algorithms. It is natural to ask: what if we put secure computation and sketches together? We investigated the question and the findings are interesting: we can get security, we can get scalability, and somewhat unexpectedly, we can also get differential privacy – for free. Our study started from building a secure computation protocol based on the Flajolet-Martin (FM) sketches, for solving the Private Distributed Cardinality Estimation (PDCE) problem, which is a fundamental problem with applications ranging from crowd tracking to network monitoring. The state of art protocol for PDCE [33] is computationally expensive and not scalable enough to cope with big data applications, which prompted us to design a better protocol. Our further analysis revealed that if the cardinality to be estimated is large enough, our protocol can achieve (ε,δ)-differential privacy automatically, without requiring any additional manipulation of the output. The result signifies a new approach for achieving differential privacy that departs from the mainstream approach (i.e. adding noise to the result). Free differential privacy can be achieved because of two reasons: secure computation minimizes information leakage, and the intrinsic estimation variance of the FM sketch makes the output of our protocol uncertain. We further show that the result is not just theoretical: the minimal cardinality for differential privacy to hold is only 10^2-10^4 for typical parameters.

[†]Changyu Dong and Zheli Liu are the corresponding authors.

1 Introduction

Data privacy has become an increasingly acute problem, especially when the hunger for data drives large-scale collection and (mis)use, without well-thought-out precautions in place. The tension between data utilization and data privacy has developed into a societal challenge and led to stricter regulations, such as HIPAA [1], GLBA [2] and GDPR [3]. The pressing need for privacy has greatly stimulated research on secure computation [18]. Secure computation allows collaborative computation over private datasets held by multiple mutually untrusted parties, without revealing any information except what can be inferred from the output. Thus, secure computation has been regarded as one of the key privacy enhancing technologies [4].

While many secure computation protocols have been proposed to carry out various data processing tasks in a privacy preserving fashion, their scalability is often open to doubt. Despite the fact that the efficiency of secure computation has been drastically improved, secure computation is still orders of magnitude slower than computation in the clear. The overhead might be acceptable if the data to be processed is small, but it can be prohibitive when the data is big. Yet, the "killer" applications of secure computation are often data-intensive, and this has become a major impediment to the widespread use of secure computation.

One good example is *Private Distributed Cardinality Estimation* (PDCE). Cardinality estimation, the task of determining the number of distinct elements in the union of multiple sets, is of particular importance in databases, data mining and distributed systems [5, 37, 38, 67]. While the task is easy to perform when data is in a single small database, it becomes challenging when data is collected independently from multiple sources at a high rate [52]. Naively maintaining a counter at each source and summing the counters up will not work because more often than not, there are duplicates in the data being collected. The task is even more challenging if privacy is needed. PDCE has numerous applications, for example:

- **Scientific research and user studies**. Surveys and questionnaires are commonly used in medical science, social science and business studies to help researchers discover interesting correlations (e.g. [56]). It is not uncommon that several organizations independently collect data through surveys and questionnaires over the same population. For example, diet habits could be surveyed by researchers from a medical institution, a government agency, and an insurance company, for different projects. If pooled together, the data would be of a much higher utility and could lead to improved decision making. For instance, the number of distinct individuals across all datasets having a certain diet habit could help identifying risk factors related to chronic diseases such as diabetes, while each individual dataset may be too small to draw a convincing conclusion. However, the data cannot be pooled together in practice because of privacy obligations imposed on each data collector.

- **Crowd counting and tracking** [66]. The task of estimating the number of individuals entering or passing by a place is key in urban planning, surveillance, public health study and retail analytics, to understand the effectiveness of building and road design, the patterns of human mobility, the spread of infectious diseases, and the patterns of customer behavior. Many existing commercial systems identify and track people though personally identifiable information (PII), such as fingerprints of mobile devices [11] or MAC addresses of WiFi cards [51,53], collected through a distributed network of sensors or WiFi hot-spots that are deployed e.g. in a big shopping mall or a retail chain across the country [62]. Usually, to obtain the estimate, the data is transmitted to, stored and processed in a central database. However, this has already raised widespread privacy concerns [63, 64]. Ideally the estimate should be obtained without the need to store or transmit PII. Also, if published, it should not disclose information about any specific individual.

- **Network monitoring and statistics**. An example is the detection of DDoS attacks by collecting information at an ISP's border routers and identifying sudden increases in the total number of distinct source IP addresses. This detection method works because the attacker usually commands many "zombies" distributed across the Internet to send packets with randomly spoofed IP source addresses to the victim [50]. Another example is that many websites nowadays use Content Distribution Networks to provide load balancing and fast access. One statistic that web masters often want to estimate is the total number, across all replicas, of distinct visitors who accessed the website [7]. This can be reduced to the distributed cardinality estimation problem. In both examples, privacy concerns are raised due to the fact that IP addresses can be used to track back users, revealing confidential information ranging from their location to personal and behavioral traits. A second concern, posed by the scale of the Internet, is to be able to address the distributed cardinality estimation problem efficiently.

Driven by the need, PDCE has attracted substantial interest [10,19,23,26,30,31,33,41,47,61,65]. The current state of the art is a secure computation protocol proposed by Fenske et.al. in CCS'17 [33]. Functionality and privacy-wise, the protocol is impeccable. However, it is not scalable enough, because it relies on expensive public key encryption and computationally demanding sub-protocols such as verifiable shuffling. The running time of the protocol is in the order of hours when the cardinality is in the order of 10^4. The protocol can fulfill its task for measuring the number of visitors to the Tor network because the cardinality to be estimated is small. However, it cannot cope with mainstream big data applications, involving million or billion sized sets, because the protocol would require weeks or years to finish.

In the data mining community, the use of sketches has gained popularity recently [17]. Sketches are space-efficient data structures that summarize massive data, so that it can be efficiently processed, stored, and queried. Sketches allow representing data in sub-linear or constant space, and thus can be employed to improve the efficiency and scalability of algorithms. Sketches are lossy and do not preserve all the information in the data they represent. Thus, sketch-based computation returns only approximate answers. That said, big data applications often do not require exact answers and the parameters of sketches can often be adjusted to obtain sufficiently accurate answers. Due to the use of sketches, many real world systems can keep up with exponentially increasing data (e.g. [6,40]).

Contributions In this paper, we build and analyse a new secure computation protocol based on the Flajolet-Martin (FM) sketch [34], for solving the PDCE problem. Initially, our intention was to make PDCE more practical. The protocol fulfils this intention very well. As expected, the protocol achieves high security, as well as much better efficiency and scalability than the state of the art [33]. Yet, this is not the end of the story. In the study we also found that the combination of secure computation and the FM sketch allows us to obtain differential privacy at no extra cost. This is interesting because differential privacy is a much desired property in PDCE, and neither secure computation nor the FM sketch provides it on its own. In more detail, the protocol has the following important features:

- **Highly Secure** Similar to exisiting work [30,31,33,41,47], we consider the scenario where a set of Data Parties (DPs) collect data and want to use some *untrusted* Computation Parties (CPs) to aggregate the data and estimate cardinality.

In such a setting, the untrusted CPs are modelled as corrupted and controlled by a single adversary. Unlike the vast majority of previous work that considers only semi-honest CPs, our protocol is developed on top of the SPDZ framework [22], and thus is secure in the presence of malicious CPs that can behave arbitrarily. For the total of d CPs, our protocol can tolerate up to $d-1$ corrupted malicious CPs. Our protocol can achieve the following security goals as long as there exists one honest CP: (1) the adversary learns nothing from executing the protocol except the output of the protocol; (2) the adversary cannot affect the correctness of the computation without being detected. We formally prove the security of the protocol in the UC model [14].

- **Efficient and scalable**. We design our protocol around the FM sketch. By using FM sketches, we can accurately estimate the cardinality, while reducing the complexity of our protocol to logarithmic (in the maximum cardinality to be estimated). This is in contrast to the majority of the existing protocols [10, 19, 23, 30, 31, 33, 41, 47, 61] whose complexity is linear. Also, we reduce much of the computation needed for the online phase by using offline pre-processing. As a comparison, the protocol in [33] needs almost 1 hour in LAN to estimate the cardinality of a set containing **30,000** elements; while our protocol only needs less than 50 minutes (in WAN) to estimate the cardinality of a set containing **1 billion** elements, and the online phase running time is only about 5 seconds.

- **Offers differential privacy for free**. The most interesting finding from our study is that our protocol can achieve differential privacy [28] for free (i.e. without the need to add noise and/or further manipulate the output). In the security models for secure computation, the adversary is allowed to infer information from the output of the protocol. This sometimes is inadequate because individuals may still be re-identified through such inference. It is often desirable in applications like PDCE to additionally disallow such inference attacks by making the output from the protocol differentially private. We proved that, given the privacy parameters (ε, δ) and FM sketch parameters, if the cardinality to be estimated is sufficiently large, then the estimated cardinality output from our protocol satisfies (ε, δ)-differential privacy. What makes the finding so interesting is that neither secure computation nor sketches provide differential privacy on their own. However, we showed for the first time that when we put the two together, they complement each other by providing something the other lacks. The intrinsic estimation variance of FM sketches now makes the output of secure computation uncertain, thus can substitute the noise we usually need to add in order to achieve differential privacy. Secure computation makes it possible to do the computation without revealing anything except the output, which means the sketches are now hidden and any information leaked by the sketches is now concealed. As a consequence, differential privacy can be achieved. We further show that this is not just a theoretical result. The lower bound of the cardinality for differential privacy to hold is reasonably small. Given typical parameters, the lower bound is usually only $10^2 - 10^4$. Thus, differential privacy can be easily satisfied in real world applications with our protocol. The technique used in our analysis is quite general, thus we would expect that with some modifications, it could be applied to other sketch based secure computation protocols as well. As the last remark, existing PDCE protocols [31, 33, 41, 47, 61], achieve differential privacy by adding noise, which however incurs a cost. This is especially true in [33], in which a large portion of the computation is spent on encrypting a large number $(10^4 - 10^5)$ of noise bits and shuffling them with the data. Therefore, free differential privacy is beneficial to the efficiency and scalability of our protocol as well.

2 Related Work

In the literature, several PDCE protocols are also called Private Set Union Cardinality (PSU-CA) protocols [19, 23, 26, 30, 33]. However, the original definition of PSU-CA [19] requires the output to be the *exact* cardinality, while quite a few protocols [26, 30, 33] output an *estimate* close to the exact cardinality. To avoid confusion, we use the term PDCE in this paper and regard PSU-CA as a special case (in which the estimation error is 0). Note that not outputting the exact cardinality is not necessarily a deficiency. When differential privacy is required, the output anyway cannot be exact.

There are two different flavours of PDCE protocols: the first is that the DPs collect and compute the cardinality, without using CPs; the second is that the DPs only collect the data, and the CPs compute the estimation. We call the former DP-PDCE and the latter CP-PDCE to differentiate them. Our protocol is a CP-PDCE protocol. One approach [19, 23] for DP-PDCE is to reduce it to a Private Set Intersection Cardinality (PSI-CA) problem. The cardinality of union can be obtained by using the inclusion-exclusion principle. However, the inclusion-exclusion principle leads to exponential complexity (in the number of sets), therefore those protocols are limited to the two-party case. There are a few DP-PDCE [23] and CP-PDCE [10, 30] protocols based on Bloom Filters. The protocol in [10] is not secure, and [30] proposed a more secure variant of the protocol. The protocol in [23], as mentioned earlier, uses the inclusion-exclusion principle, thus is not scalable. All the above protocols have computational and communication complexity linear in the maximum cardinality to be estimated. FM sketches were used by the DP-PDCE protocol in [26] to lower its complexity to logarithmic. However, only a two-party protocol was given in the paper with a brief statement that a multiparty protocol is feasible. The DP-PDCE protocol in [65] also uses FM sketches. However this protocol is not secure. The protocol reveals more information than the cardinality itself because the parties learn the union sketch in the protocol. It also assumes none of the parties collude, which is a very strong assumption. None of the aforementioned protocols supports differential privacy. There are protocols that provide differential privacy [31,33,41,47,61]. The CP-PDCE protocols in [31,33,41,47] were all designed for gathering statistics in the Tor network [25], which naturally requires a high degree of privacy as the aim of Tor is to keep users anonymous. The protocols in [31,41,47] consolidate the observations of each DP into a counter, thus cannot eliminate duplicates when the counters are aggregated together. In [33], each DP maintains a hash-table with a public hash function for the observations. If an observation occurs multiple times, regardless by the same DP or by different DPs, it will be hashed into the same bin of the hash-table and the duplicates can be eliminated. However, in order to reduce collisions and maintain a reasonable accuracy, the hash-table size needs to be much larger than the maximum cardinality to be estimated. This impacts the efficiency and scalability of the protocol significantly. In [61], each DP represents its observations as a bit vector, enforces differential privacy on the vector using randomized response, and then passes the vector to a CP who can estimate cardinality of the set union. The estimation has a high standard deviation (in the order of the size of the universe of the set), thus the result is not accurate enough for many applications. All the above protocols except [33] consider the semi-honest or an even weaker adversary model, mainly for efficiency reasons, while our protocol and [33] are secure against more powerful malicious adversaries.

There is a large body of research works on Private Data Aggregation in which multiple data collectors (DPs) and data aggregators (CPs) are involved in aggregating data and outputting some statistics. Some works consider a much weaker security model and assume a trusted aggregator, who aggregates data from the DPs in plaintext and then adds noise before outputting the result [49,60]. There are protocols that consider an untrusted aggregator, e.g. for computing private sum [16,55,59], or for frequency estimation over categorical data [16,32], or for computing KNN and median [48]. Sketches (e.g. Count and Count-min sketches) were used in [48,49] to make the protocols more efficient.

3 Preliminaries

3.1 Flajolet-Martin (FM) Sketches

We briefly review FM sketches. More details and analysis can be found in [26,34,57]. An FM sketch is a probabilistic data structure for counting the number of distinct elements in a multi-set. The data structure is a w-bit binary vector. Let FS denote an FM sketch, and $\mathsf{FS}[i]$ $(0 \le i \le w-1)$ denote the ith bit in FS. An FM sketch is built using two functions:

- $H : \{0,1\}^* \rightarrow \{0,1\}^{w-1}$: a hash function that maps an input uniformly to a $(w-1)$-bit string.
- $\rho : \{0,1\}^{w-1} \rightarrow [0, w-1]$: a function that takes a $(w-1)$-bit string as input and returns the number of trailing zeroes in it.

Initially, all bits in FS are set to 0. To estimate the cardinality of a multi-set S, for each element $x \in S$, we hash x and set $\mathsf{FS}[\rho(H(x))] = 1$. The quantity N, which is the number of distinct elements in S, can be estimated using an estimator z_N that is the index of the first[1] 0 bit in FS, i.e. $\mathsf{FS}[z_N] = 0$ and $\forall 0 \le j < z_N$, $\mathsf{FS}[j] = 1$. The expected value of z_N is close to $\log(\phi N)$, where $\phi = 0.77351$ is a correction factor. Therefore, N is roughly $2^{z_N}/\phi$. It is clear that the size of the sketch w must be larger than $\log(\phi N)$, otherwise z_N might not be correct. As suggested in [34], $w \ge \log(N) + 4$ should suffice.

The standard deviation of z_N is 1.12, which is too high (i.e. an estimation using z_N will typically be one binary order of magnitude off the true cardinality). To remedy this problem, [34] suggested to use m sketches, each with an independent hash function. Then we can obtain m estimators $z_{N,1}, \ldots, z_{N,m}$, sum them to $Z_N = z_{N,1} + \ldots + z_{N,m}$, and use the average $\frac{Z_N}{m}$ to estimate the cardinality N. The standard deviation of Z_N is $1.12 \cdot \sqrt{m}$. Thus, the standard deviation of $\frac{Z_N}{m}$ is $\frac{1.12}{\sqrt{m}}$, which is much smaller. In [57], the authors suggested the following, modified formula that can achieve better estimation accuracy:

$$\widetilde{N} = \frac{2^{\frac{Z_N}{m}} - 2^{-\kappa \cdot \frac{Z_N}{m}}}{\phi} \tag{1}$$

where $\widetilde{N}$ is the cardinality estimated from m sketches, and $\kappa = 1.75$ is a correcting factor. In [26], it was shown that the accuracy of the estimation can be improved by enlarging m. This implies that the accuracy of the estimation can be adjusted to the desired level, by choosing a suitable m.

An important property of FM sketches that we use in the design of our protocol is that they can be merged. If we have two FM sketches FS_1 and FS_2 built with the same hash function, but on different sets S_1 and S_2 respectively, then bit-wisely ORing the two sketches produces a new FM sketch $\mathsf{FS}_\cup$ that counts the union of the two sets S_1 and S_2. This process is lossless: $\mathsf{FS}_\cup$ is exactly the same as the sketch built using the union from the scratch. This holds also in the case of more than two sketches. Our protocol will use this property to union FM sketches from different DPs.

3.2 SPDZ

In this section, we briefly review the SPDZ scheme [21,22,43,44] that will be used as the underlying framework for our protocol. We will follow mostly the notations in [43,44]. Essentially, SPDZ is a secret-sharing based multiparty computation (MPC) scheme that supports secure computation over a

[1]We use the most significant bit first ordering throughout the paper.

finite field (e.g. $\mathbb{F}_p$ for some prime p). One notable feature of SPDZ is its 2-phase design: there is a *pre-processing phase* that produces correlated random values that are independent of the task to be securely computed, and the pre-computed random values will then be consumed in the *online phase* to enable very efficient computation. SPDZ aims to provide highly efficient online phase primitives such as secure addition and secure multiplication. Then high-level protocols can be implemented on top of SPDZ by calling the online phase primitives to compute a task expressed as an arithmetic circuit. In addition to efficiency, another benefit that SPDZ offers is strong security: it is UC secure against a static, active adversary corrupting up to $n-1$ parties, and this strong security extends to high level protocols implemented on top of it.

On the technical side, SPDZ utilizes authenticated shares. In SPDZ, a value $x \in Z_p$ in the shared form is defined as:

$$[\![x]\!] = (x_1, \cdots, x_n, m_1^{(x)}, \cdots, m_n^{(x)}, \Delta_1, \cdots, \Delta_n),$$

and each party P_i holds a tuple $[\![x]\!]_i = (x_i, m_i^{(x)}, \Delta_i)$ such that:

$$x = \sum_{i=1}^{n} x_i, \quad m^{(x)} = \sum_{i=1}^{n} m_i^{(x)}, \quad \Delta = \sum_{i=1}^{n} \Delta_i.$$

Each value is authenticated by a MAC. In the above, Δ is a global MAC key and the MAC is $m^{(x)} = x \cdot \Delta$. The authenticity of x can be verified by letting each P_i compute $\sigma_i = m_i^{(x)} - x \cdot \Delta_i$ and broadcast σ_i, then check if $\sum_{i=1}^{n} \sigma_i = 0$. The three parts in the tuple $[\![x]\!]_i$ are additive shares of x, the MAC and the MAC key respectively.

In our protocols, we will explicitly use the following online phase primitives from SPDZ:

- $[\![x+y]\!] \leftarrow [\![x]\!] + [\![y]\!]$: given shared values $[\![x]\!]$ and $[\![y]\!]$, compute the sum. This is done locally by each party P_i by computing $[\![x+y]\!]_i = (x_i + y_i, m_i^{(x)} + m_i^{(y)}, \Delta_i)$.
- $[\![a+x]\!] \leftarrow a + [\![x]\!]$: add a shared value $[\![x]\!]$ with a public value a. To do so, P_1 computes $[\![a+x]\!]_1 = (x_1 + a, m_1 + a \cdot \Delta_1, \Delta_1)$, and each other party P_i computes $[\![a+x]\!]_i = (x_i, m_i + a \cdot \Delta_i, \Delta_i)$.
- $[\![a \cdot x]\!] \leftarrow a \cdot [\![x]\!]$: multiply a shared value $[\![x]\!]$ with a public value a. Each P_i computes locally $[\![a \cdot x]\!]_i = (a \cdot x_i, a \cdot m_i, \Delta_i)$ from $[\![x]\!]_i$.
- reveal($[\![x]\!]$): reveal x in a shared value $[\![x]\!]$, each P_i broadcasts x_i in $[\![x]\!]_i$ and computes $x = \sum_{i=1}^{n} x_i$.
- $[\![x \cdot y]\!] \leftarrow [\![x]\!] \cdot [\![y]\!]$: multiply two shared values. It is done by using Beaver's triple [12], i.e. a triple $([\![a]\!], [\![b]\!], [\![c]\!])$ where a, b are random numbers in $\mathbb{F}_p$ and $c = a \cdot b$. The triples are generated in the pre-processing phase. In the online phase when computing multiplication, a fresh random triple is used. It works by revealing (which requires broadcast) $[\![\varepsilon]\!]$ and $[\![\rho]\!]$ where $[\![\varepsilon]\!] \leftarrow [\![x]\!] - [\![a]\!]$ and $[\![\rho]\!] \leftarrow [\![y]\!] - [\![b]\!]$. Then the product can be obtained as $[\![x \cdot y]\!] \leftarrow [\![c]\!] + \varepsilon [\![b]\!] + \rho [\![a]\!] + \varepsilon\rho$.
- Output($[\![x]\!]$): this is used at the end of a protocol to output the final result x. It first checks the MACs of all values previously revealed in the protocol. If it fails, then aborts. Otherwise, it reveals x in $[\![x]\!]$ to all parties, and checks the MAC of x. It aborts if it fails, and it outputs x otherwise.

Our protocols will use the pre-processing protocols in SPDZ for generating Beaver's triples. Since pre-processing is necessary for our protocols, we will treat the pre-processing phase as in place implicitly and not explicitly mention calling it, in the description of the protocols.

In SPDZ (and in many other secret-sharing MPC schemes), since computation over shares is simple modular addition and multiplication in a small finite field, the performance bottleneck of online protocols is often network communication [43, 44]. Therefore, reducing the number of rounds and number of interactions is crucial to the efficiency of the online protocols.

3.3 Differential Privacy

Differential privacy [27] is a well-established principle that quantifies the privacy impact on individuals, when their private information is included in a dataset and some statistics obtained from the dataset are released. The first definition of differential privacy is the following:

Definition 1 (ε-differential privacy [27]). *A randomized mechanism $f : \mathcal{D} \to \mathcal{R}$ gives $\varepsilon-$differential privacy, where ε is a positive real number, if for all data sets D_1 and D_2 differing in at most one element, and all $R \subseteq \mathcal{R}$,*

$$e^{-\varepsilon} \cdot Pr[f(D_2) \in R] \leq Pr[f(D_1) \in R] \leq e^{\varepsilon} \cdot Pr[f(D_2) \in R].$$

Definition 1 is very strong but also often renders the output unusable, since it incurs substantial distortion to be enforced. Therefore, (ε, δ)-differential privacy is often used:

Definition 2 ((ε, δ)-differential privacy [28]). *A randomized mechanism $f : \mathcal{D} \to \mathcal{R}$ gives (ε, δ)-differential privacy, where (ε, δ) are positive real numbers, if for all data sets D_1 and D_2 differing in at most one element, and all $R \subseteq \mathcal{R}$,*

$$e^{-\varepsilon} \cdot Pr[f(D_2) \in R] - \frac{\delta}{e^{\varepsilon}} \leq Pr[f(D_1) \in R] \leq e^{\varepsilon} \cdot Pr[f(D_2) \in R] + \delta.$$

Intuitively, (ε, δ)-differential privacy ensures that for all adjacent D_1, D_2, the absolute value of the privacy loss will be bounded by ε with a probability at least $1 - \delta$.

3.4 Statistical Security

We briefly review the notion of statistical security [35] that we use in our ZeroTest sub-protocol (see Section 4.5). This notion requires that the views of protocol execution can be simulated such that the distributions of real and simulated views are statistically indistinguishable. Formally, let X and Y be distributions with finite sample spaces V and W and $\Delta(X,Y) = \frac{1}{2}\sum_{v \in V \cup W} |Pr(X = v) - Pr(Y = v)|$ the statistical distance between them. We say that the distributions are statistically indistinguishable if $\Delta(X,Y) \leq \mathsf{negl}(\lambda)$ where negl is a negligible function and λ is some statistical security parameter. As usual, a function is negligible if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $\mathsf{negl}(n) < \frac{1}{p(n)}$. Statistical security is information theoretic, i.e. it holds even if the adversary has unbounded computational power. The statistical security parameter usually can be smaller than the computational security parameter (e.g. 40 is often used in the literature [39, 54]).

3.5 Universal Composability (UC)

We briefly review the UC framework [14] that we use to prove the security of our protocol. Being UC secure means that our protocol can be freely composed with other protocols and still be secure. The UC framework is defined in terms of comparing a real world execution and the execution in an ideal world, in the presence of an adversary (environment). Security in UC is defined in terms of the adversary's inability to distinguish whether it is interacting with the real protocol Π, or with a simulator in the ideal world which has access to an ideal functionality $\mathcal{F}$. If so, then we say that the protocol Π securely realizes the functionality $\mathcal{F}$. Intuitively, the ideal world is secure by definition, and a successful simulation means that the adversary running the protocol in real world cannot do more damage than what is allowed in the ideal world, hence the protocol is secure.

Let the adversary be $\mathcal{Z}$. In the beginning of an execution, $\mathcal{Z}$ chooses inputs for all parties and gets their outputs when the execution finishes. It also controls some corrupted parties, which means $\mathcal{Z}$ will instruct what they should do during the execution and see the communication and internal states of them. When $\mathcal{Z}$ stops, it outputs a bit. Security is established by showing the existence of a simulator $\mathcal{S}$ that interacts with both $\mathcal{F}$ and $\mathcal{Z}$. The simulator should be able to simulate the view of the protocol that looks like what $\mathcal{Z}$ would see in a real attack by playing the honest parties' role when interacting with $\mathcal{Z}$, but without access to the input and state of the honest parties. One significant difference in the simulation in UC and in stand-alone environment is that $\mathcal{Z}$ can query the corrupted parties during the execution (rather than just collect the views after the execution). This means some techniques such as rewinding cannot be used in UC proofs. For a more formal and complete account of the UC framework, please refer to [14].

4 The PDCE Protocol

4.1 Overview

In the PDCE protocol, we have a set of n honest Data Parties (DPs) and a set of d untrusted (up to $d-1$ can be malicious) Computation Parties (CPs). The DPs are responsible for data collection. They observe the events of interest, e.g. IP addresses of the visitors, and record them locally as a set of FM sketches. After the data has been collected, the DPs secret-share the sketches among the CPs, who will securely combine them, and compute the estimator Z_N of the count of distinct values. The protocol has four phases: initialization phase, offline phase, data collection phase, and data aggregation phase. Each phase involves certain sub-protocols.

4.2 Initialization Phase

In this phase, the parties negotiate parameters to be used in the protocol. This phase only needs to run once when setting up the system. Firstly, all parties need to agree on a finite field $\mathbb{F}_p$. This field will be used as the basis of data representation, secret sharing and all computation. The modulus p is decided by three parameters: (1) λ, which is a statistical security parameter (e.g. 40); (2) τ, which determines the size of the plaintext domain (integers between $[0, 2^\tau - 1]$); (3) M, e.g. 32768, which comes from the BGV somewhat homomorphic encryption [13] used by SPDZ. Specifically, the parties choose p that is a $(\lambda+\tau)$-bit prime number and M divides $p-1$. Next, the parties agree on the parameters for FM sketches. Given the accuracy and privacy requirements, they decide m (the number of sketches to be used). Based on the pre-knowledge of the maximum number of items that can be observed collectively, the parties decide w (the size of each sketch). Finally, the CPs run the setup protocol of SPDZ to obtain the parameters and keys for SPDZ.

4.3 Offline Phase

In the offline phase, the CPs run the pre-processing protocol of SPDZ. In addition, they also run a few other offline protocols to generate various random values that will be used later in the data collection and aggregation phases. The offline protocols we use already exist in the literature, therefore we only give a high level description of them here. The protocol details and references can be found in Appendix B.

- Rand(): generates $[\![r]\!]$, the shares of a random value $r \in_R \mathbb{F}_p$.
- Rand2(): generates $[\![b]\!]$, the shares of a random bit $b \in_R \{0,1\}$.
- RandExp(l): generates $([\![R^{-1}]\!], [\![R]\!], [\![R^2]\!], \ldots, [\![R^l]\!])$, the shares of a random number $R \in_R Z_p^*$, as well as the shares of its ith powers (for $i = -1$ and $2 \leq i \leq l$).

4.4 Data Collection Phase

At the beginning of this phase, the DPs choose a keyed hash function H, a pseudorandom function PRF, and establish a secret key sk for PRF among them. The secret key sk can be established using an authenticated group key exchange protocol (e.g. [42]). The PRF and the key sk will be used for deriving hash keys, so that m independent FM sketches can be constructed using H and different hash keys. For $1 \leq j \leq m$, the jth hash key is $k_j = PRF(sk, j)$. Then each DP maintains m FM sketches, observes items and adds them into its FM sketches. At the end of this phase, each DP splits its FM sketches into secret shared form, and sends the shares to the CPs. The protocol for data collection is shown in Protocol 1, and the sub-protocol $Share(x)$ is shown in Protocol 2.

Protocol 1: Data Collection

```
Input: Each DP's input is $sk$, the shared key for the PRF
Result: The CPs obtain the shares of the FM sketches
// Initialize FM sketches
1  Each $DP_i$ initialize $m$ FM sketches, each is $w$-bit
// Collect data
2  Whenever $DP_i$ observes an item $o$, it does the following:
       // add $o$ to sketch (see Sec. 3.1)
3      for $j = 1; j \leq m; j{+}{+}$ do
4          Compute $l = \rho(H(k_j||o))$;
5          Set $\mathsf{FS}_i^j[l] = 1$;
6      end
// Finish data collection
7  After data has been collected, each $DP_i$ does the following:
8      for $j = 1; j \leq m; j{+}{+}$ do
9          for $l = 0; l \leq w-1; l{+}{+}$ do
10             Run $Share(\mathsf{FS}_i^j[l])$ with the CPs;
11         end
12     end
```

Protocol 2: $Share(x)$

```
Offline: CPs run $[\![a]\!] \leftarrow Rand()$, where $a \in_R \mathbb{F}_p$.
Input: The DP's input is $x$, the value to be shared.
Result: The CPs obtain $[\![x]\!]$
1  CPs reveal $a$ to DP;
2  DP computes $x - a$ and broadcasts it to all CPs;
3  CPs obtain $[\![x]\!] = [\![a]\!] + (x - a)$;
```

4.5 Data Aggregation Phase

This phase involves only the CPs. The CPs first merge the shares from the DPs into m shared FM sketches such that each slot in the sketches holds either a zero or a positive integer. Then, they convert the integer FM sketches into binary FM sketches. After that, they extract the estimator Z_N from the sketches, and compute the count from the estimator locally.

Merge Shares At the start of the data aggregation phase, each CP holds the shares of all the FM sketches from all DPs. The first step for each CP is to merge the shares of the sketches to get the shares of a set of m (integer) FM sketches that record the union of observations from all DPs. As mentioned in Section 3.1, merging FM sketches can be done by bit-wisely ORing the sketches. However, the Boolean OR operation corresponds to multiplication of shared values. A naive implementation of this step thus would require $(n-1)\cdot m \cdot w$ multiplication operations and thus $(n-1)\cdot m \cdot w$ rounds of communication, where n is the number of DPs, m is the number of FM sketches generated by each DP, and w is the bit-size of the FM sketches. To reduce the cost, in our protocol, we merge the shares by addition. The protocol is shown in Protocol 3. For the l-th bit in the j-th FM sketches, the CPs locally sum up the n shares for that bit from all DPs. At the end, the CPs obtains the shares of m integer FM sketches such that 0 in the integer FM sketches corresponds to 0 in binary FM sketches, and non-zero corresponds to 1. The integer FM sketches will be converted to binary sketches in the next step. The only operation needed in this step is addition. Thus, no interaction is required. Looking ahead, the next step requires in total $2 \cdot m \cdot w$ rounds of interaction, thus the total cost is much less than the naive implementation in real applications where the number of DP is often large.

Protocol 3: MergeShares

```
Input: Each $CP_k$ holds $[\![\mathsf{FS}_i^j[l]]\!]_k$
       $(1 \leq i \leq n, 1 \leq j \leq m, 0 \leq l \leq w-1)$
Result: $[\![\mathsf{FS}_\cup^j[l]]\!]_k$ $(1 \leq j \leq m, 0 \leq l \leq w-1)$
1  for $j = 1; j \leq m; j{+}{+}$ do
2      for $l = 0; l \leq w-1; l{+}{+}$ do
3          $[\![\mathsf{FS}_\cup^j[l]]\!]_k = \sum_{i=1}^n ([\![\mathsf{FS}_i^j[l]]\!]_k)$;
4      end
5  end
```

Protocol 4: $ToBinary([\![\mathsf{FS}_\cup^1[0]]\!], \cdots, [\![\mathsf{FS}_\cup^1[w-1]]\!], \cdots, [\![\mathsf{FS}_\cup^m[0]]\!], \cdots, [\![\mathsf{FS}_\cup^m[w-1]]\!])$

```
Input: $[\![\mathsf{FS}_\cup^j[l]]\!]$ $(1 \leq j \leq m, 0 \leq l \leq w-1)$, shares of the $m$
       integer FM sketches.
Result: $[\![\mathsf{BFS}_\cup^j[l]]\!]$ $(1 \leq j \leq m, 0 \leq l \leq w-1)$, shares of the
        $m$ converted binary FM sketches.
1  for $j = 1; j \leq m; j{+}{+}$ do
2      for $l = 0; l \leq w-1; l{+}{+}$ do
3          $[\![\mathsf{BFS}_\cup^j[l]]\!] = ZeroTest([\![\mathsf{FS}_\cup^j[l]]\!])$;
4      end
5  end
```

Convert to Binary Sketches As shown in Protocol 4, the second step is to covert each $\mathsf{FS}_\cup^j$ back to the normal binary FM sketches[2], so that we can later extract the estimator from them. This is done by running a zero test protocol among the CPs on each slot that sets the slot to 0 if the value stored in it is 0, or to 1 otherwise.

Here we use the protocol from [46]. The protocol is based on the following idea: to test whether a is 0 or not, we first

[2]To clarify, here binary means $\{0,1\}$ in $\mathbb{F}_p$, not $\{0,1\}$ in $\mathbb{F}_2$

Protocol 5: *ZeroTest*($[\![a]\!]$)

```
Offline:
    for $i = 0, \cdots, l-2$, where $l$ is the bit length of $p$ do
        $[\![r_i]\!] \leftarrow Rand2()$;
    end
    $[\![r]\!] \leftarrow \sum_{i=0}^{l-2} 2^{l-2-i}[\![r_i]\!]$;
    // interpolate the lookup polynomial
    $(\tau, \beta_0, \cdots, \beta_\tau) \leftarrow interpolate()$;
Input: $[\![a]\!]$, where $a$ is a $\tau$-bit integer.
Result: $[\![b]\!]$, where $b = 0$ if $a = 0$, $b = 1$ otherwise
1 $[\![m]\!] = [\![r]\!] + [\![a]\!]$;
2 Reveal $[\![m]\!]$;
3 $[\![1+h]\!] = 1 + \sum_{i=l-1}^{l-\tau}([\![r_i]\!] + m_i - 2[\![r_i]\!] \cdot m_i)$;
4 $[\![b]\!] = Lookup([\![1+h]\!], \tau, \beta_0, \cdots, \beta_\tau)$
```

Protocol 6: *Lookup*($[\![x]\!], \ell, \beta_0, \ldots, \beta_\ell$)

```
Offline: $([\![R^{-1}]\!], [\![R]\!], [\![R^2]\!], \cdots [\![R^\ell]\!]) \leftarrow RandExp(\ell)$;
Input: $[\![x]\!]$, where $x$ is an integer; $\ell$ is the degree of the
    lookup polynomial $f(\cdot)$; $\beta_0, \ldots, \beta_\ell$ are the coefficient
    of $f(\cdot)$.
Result: $[\![y]\!]$, where $y = f(x)$.
1 $[\![a]\!] = [\![R^{-1}]\!] \cdot [\![x]\!]$;
2 Reveal $[\![a]\!]$;
3 for $i = 2, \cdots, \ell$ do
4     $[\![x^i]\!] = a^i \cdot [\![R^i]\!]$
5 end
6 $[\![b]\!] = \sum_{i=0}^{\ell} \beta_i \cdot [\![x^i]\!]$
```

compute $r+a$ where r is a random integer, and then compute the Hamming distance h between $r+a$ and r. Obviously, if $a=0$, then $h=0$; otherwise h is a small integer in $[1,\tau]$, where τ is the bit length of the plaintext. As h is small, it is feasible to use a lookup function that is a polynomial $f(\cdot)$ such that $f(0)=0$ and $f(x)=1$ for all other $x \in [1,\tau]$. There is a small technicality that $f(0)$ cannot be evaluated without leaking information. To see that, note that in the first line of Protocol 6, if $x=0$ then $a=0$, and revealing a will reveal whether x is 0. Thus in line 3 of Protocol 5, 1 is added to h so that the input to the polynomial will never be 0. The lookup polynomial will be interpolated accordingly (e.g. using Lagrange Interpolation), and evaluating f at $h+1$ will output 0 if h is 0 or 1 otherwise. The *ZeroTest* protocol is shown in Protocol 5, and the sub-protocol *Lookup* for evaluating the lookup function is shown in Protocol 6 (both are from [46]).

Extract Estimator Recall that given an FM sketch, one can extract z_N, i.e. the index of the first 0 bit in the sketch. When using m FM sketches, the sum $Z_N = \sum_{i=1}^{m} z_{N,i}$ will be used to estimate the number of distinct observed items as $\widetilde{N} = \frac{2^{\frac{Z_N}{m}} - 2^{-\kappa \cdot \frac{Z_N}{m}}}{\phi}$ (see Section 3.1). The formula is deterministic and invertible, therefore revealing $\widetilde{N}$ and revealing Z_N are essentially equivalent. Because of this, we can let the protocol output Z_N rather than $\widetilde{N}$ without compromising correctness or security. With Z_N, each CP can locally compute $\widetilde{N}$.

Z_N can be extracted using the following simple idea: firstly,

Protocol 7: *ExtractZ*($[\![\mathsf{BFS}_\cup^1[0]]\!], \cdots, [\![\mathsf{BFS}_\cup^1[w-1]]\!], \cdots, [\![\mathsf{BFS}_\cup^m[0]]\!], \cdots, [\![\mathsf{BFS}_\cup^m[w-1]]\!]$)

```
Input: $[\![\mathsf{BFS}_\cup^1[0]]\!], \cdots, [\![\mathsf{BFS}_\cup^1[w-1]]\!], \cdots, [\![\mathsf{BFS}_\cup^m[0]]\!], \cdots,$
    $[\![\mathsf{BFS}_\cup^m[w-1]]\!]$, the shares of the $m$ binary FM
    sketches.
Result: $Z_N$, the estimator extracted from the sketches
1 $[\![Z_N]\!] = 0$;
2 for $i = 1; i \le m; i++$ do
3     $[\![Z_N]\!] = [\![Z_N]\!] + [\![\mathsf{BFS}_\cup^i[0]]\!]$;
4 end
5 for $l = 1; l \le w-1; l++$ do
6     for $i = 1; i \le m; i++$ do
7         $[\![\mathsf{BFS}_\cup^i[l]]\!] = [\![\mathsf{BFS}_\cup^i[l-1]]\!] \cdot [\![\mathsf{BFS}_\cup^i[l]]\!]$;
8         $[\![Z_N]\!] = [\![Z_N]\!] + [\![\mathsf{BFS}_\cup^i[l]]\!]$;
9     end
10 end
11 return $Z_N \leftarrow \mathsf{Output}([\![Z_N]\!])$;
```

for each sketch, set all bits after the first 0 bit to 0, then sum up all bits in the sketch, and the result is the estimator $z_{N,i}$; then Z_N can be obtained by summing up all $z_{N,i}$'s. This is essentially what we do in Protocol 7. To set bits to 0 after the first 0 bit, the protocol does the following: for each sketch $\mathsf{FS}_\cup^i$, it sets $\mathsf{FS}_\cup^i[l] = \mathsf{FS}_\cup^i[l-1] \cdot \mathsf{FS}_\cup^i[l]$ sequentially for $1 \le l \le w-1$. By doing so, all bits before and including the first 0 bit remain unchanged, and all bits after will be set to 0 due to the chained multiplication.

Although Protocol 7 is simple, it requires $w-1$ rounds because the multiplication in each round is dependent on the output of the previous round. To improve the round efficiency, we designed another protocol (Protocol 13) that only requires $2\log(w)$ rounds. The protocol can be found in Appendix E. As we will see later in Section 6, the performance of Protocol 13 is better than Protocol 7 when the network bandwidth is limited.

Estimate the Cardinality After extracting Z_N, each CP computes the estimated cardinality locally with the formula $\widetilde{N} = \frac{2^{\frac{Z_N}{m}} - 2^{-k \cdot \frac{Z_N}{m}}}{\phi}$, as explained in Section 3.1.

5 Security Analysis

5.1 Protocol Security

We prove the security of the protocol in Section 4 in the Universally Composable framework [15]. This provides a strong notion of security and allows our protocol to serve as a component of a larger system without losing its security properties. We adopt a very strong adversary model in which the untrusted CPs are modelled as corrupted by a single adversary that is malicious, i.e. can behave arbitrarily. The adversary can corrupt all but one CPs statically. Informally, with only one honest CP, the security properties of the protocol are: (1) the adversary learns nothing from executing the protocol except the differentially private output of the protocol; (2) the adversary cannot affect the correctness of the computation without being detected. The security properties we considered in this paper are confidentiality and correctness. We leave out other properties such as robustness. In essence, the protocol will terminate if any party aborts and no result will be computed. This limitation is inherent in the underlying MPC framework we use, namely SPDZ. That said, robust MPC is an active research topic and our protocol can be migrated to a robust MPC framework when it is available.

Our adversary model is quite similar to that in [33], except that (1) in our model, the DPs are honest but in [33] they allow DPs to be corrupted adaptively; (2) In our model, malicious CPs cannot tamper with the data and the result, while in [33] a malicious CP can insert elements into the hashtable and change the result (and this cannot be prevented unless their protocol is significantly changed). Regarding whether the DPs should be assumed honest or not, we have the following remarks: (1) We model the DPs as honest mainly because, like many other differential privacy mechanisms, we need to keep the randomness, namely the PRF key, private from the adversary. Compromising this key will break the differential privacy guarantee. On the other hand, although [33] allows DPs to be corrupted, once a DP is corrupted, differential privacy guarantee is broken as well. This is because the adversary can now see the raw data collected by the corrupted DP. If the element x that differentiates D_1 and D_2 happens to be observed by the adversary, differential privacy is broken. (2) After corrupting a DP, [33] can prevent the adversary from seeing the corrupted DPs' data before corruption. This is something our protocol cannot achieve now. However, firstly [33] is used for Tor, and they consider law enforcement forcing DPs to reveal data collected a threat, but this is not common in other applications; secondly, we can easily achieve it, by secret-sharing the FM sketches when initializing them, and update them obliviously. This only adds one round of communication between DPs and CPs, and negligible computation. (3) Our DP side computation is cheap (hashing) and requires only small storage (a few MB for thousands of FM sketches and one secret key). Thus, it is relatively easy to secure DPs, e.g. using trusted hardware like Intel SGX. Spending reasonable efforts on securing DPs in exchange for much less computation on CPs seems to be a worthy trade-off.

Note, as in many proofs, we prove the security modularly in the so called $\mathcal{F}$-hybrid model. That is, we can replace an already proven secure sub-protocol with an ideal functionality. Theorem 5 states two ideal functionalities; $\mathcal{F}_{\mathsf{SPDZ}}$ and $\mathcal{F}_{\mathsf{offline}}$. The first is the ideal functionality for the SPDZ protocol, whose security has been proven in [22, 44]. The details of $\mathcal{F}_{\mathsf{SPDZ}}$ can be found in Appendix C. The second is the ideal functionality for our offline protocols. The offline protocols are from the literature, therefore we also separate them as an ideal functionality. The details of $\mathcal{F}_{\mathsf{offline}}$ as well as the full security proof (under the SPDZ framework) can be found in Appendix D. Then, the security properties of the online protocol that does the cardinality estimation are captured by an ideal functionality in Figure 1. We have the following theorem:

Theorem 1. *In the $\mathcal{F}_{\mathsf{SPDZ}}$, $\mathcal{F}_{\mathsf{offline}}$-hybrid model, the protocol in Section 4 realizes $\mathcal{F}_{\mathsf{PDCE}}$ with statistical security against any malicious adversary who statically corrupts up to $d-1$ CPs.*

The full proof of Theorem 1 is in Appendix F.

Functionality $\mathcal{F}_{\mathsf{PDCE}}$

The functionality maintains a dictionary, Val, to keep track of the authenticated values. Entries of Val lie in the (fixed) finite field $\mathbb{F}_p$ and cannot be changed, for simplicity.
Abort: On receiving **Abort** from the adversary, send $\perp$ to all parties and terminate.
Share: On receiving (share, x, id) from DP, and (share, id) from all CPs, set $\mathsf{Val}[id] \leftarrow x$.
Go: After receiving (go) from all parties, ignore messages from DP and the following methods can be called from now on.
MergeShare: On receiving $(\mathsf{mergeshare}, id^{\mathsf{FS}}, id^{\mathsf{FS}_\cup})$ from all CPs, where id^{FS} is a $(mw \times n)$ matrix and $id^{\mathsf{FS}_\cup}$ is an mw vector, all contain some ids, set for $1 \leq i \leq mw$,
$\mathsf{Val}[id_i^{\mathsf{FS}_\cup}] \leftarrow \sum_{j=1}^{n} \mathsf{Val}[id_{i,j}^{\mathsf{FS}}]$.
Lookup: On receiving $(\mathsf{lookup}, id_x, id_y, \ell, \beta_0, \cdots, \beta_\ell)$ from all CPs, check that $\ell, \beta_0, \cdots, \beta_\ell$ defines a lookup polynomial as expected, then set $\mathsf{Val}[id_y] \leftarrow \sum_{i=0}^{\ell} \beta_i \cdot (\mathsf{Val}[id_x])^i$.
ZeroTest: On receiving $(\mathsf{zerotest}, id_a, id_b)$ from all CPs, if $\mathsf{Val}[id_a] = 0$, set $\mathsf{Val}[id_b] \leftarrow 0$, otherwise set $\mathsf{Val}[id_b] \leftarrow 1$.
ExtractZ: On receiving $(\mathsf{extractZ}, id_0^1, \cdots, id_{w-1}^1, id_0^2, \cdots, id_{w-1}^2, \cdots, id_0^m, \cdots, id_{w-1}^m, id_{Z_N})$ from all CPs, count from the beginning the number of continuous 1 in $(\mathsf{Val}[id_0^i], \cdots, \mathsf{Val}[id_{w-1}^i])$ to get $z_{N,i}$, then compute $Z_N = \sum_{i=1}^{m} z_{N,i}$, set $\mathsf{Val}[id_{Z_N}] \leftarrow Z_N$.

Figure 1: Ideal Functionality for the PDCE Protocol

5.2 Differential Privacy

In this section, we will show that if the cardinality to be estimated by the FM sketches is large enough (larger than a threshold N_0), then our protocol in Section 4 satisfies (ε, δ)-differential privacy automatically, without requiring any further manipulation of the output. We noticed that in [24], the authors conclude that cardinality estimation by sketches does not preserve privacy. However, our positive result does not contradict their negative result. The reason is that in their model, the adversary can access the sketches and the final estimation result; while in our model, since MPC is used, the adversary can only access the final estimation result. The sketches are secret-shared in the protocol and are never revealed (if at least one CPs is honest). In fact, the mitigation strategies proposed in [24] are about restricting the adversary's access to the sketches, which is in line with what we do.

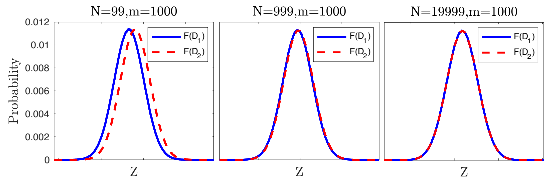


Figure 2: Results from the Monte Carlo Simulations

5.2.1 Intuition

The notion of (ε,δ)-differential privacy requires that the outputs from a randomized mechanism on two neighboring datasets should be close enough with high probability.

To start with, our protocol can be viewed as a randomized mechanism $F : 2^U \rightarrow \mathbb{Z}$. Let U be the universe of elements and $D \subseteq U$ a set comprised of the union of the observations of all n DPs. F takes D as input, internally builds m FM sketches of D, and outputs the random variable $Z_N = z_{N,1} + \cdots + z_{N,m}$. Note that F is a randomized mechanism because a randomly chosen key is used in our protocol and is renegotiated in each run. The random key is used to derive the hash keys for each FM sketch. The use of hash keys also ensures the independence of $z_{N,i}$'s, albeit they might be generated on correlated data. To see that, for H that is modeled as a perfect random function, $Pr[H(k_i||x) = y]$ is uniform and independent of $Pr[H(k_j||x) = y']$. The independence of the hash output then implies $z_{N,i}$'s are independent. Also, as we mentioned in Section 3.1, FM sketches can eliminate duplicates in data because the same element will end up with the same hash value when hashed under the same hash key, thus multiple copies of the same element will be counted as one. Although data is collected by individual DPs, we can think the final result is about the union set of the elements from all DPs. We can model F just with one input D that is the union of the n sets from the DPs.

The output Z_N from F is a random variable which can take larger values as the cardinality of D increases. Intuitively, as D becomes larger, each element in D has a smaller contribution to Z_N. Eventually, the contribution becomes so insignificant and each element's presence will have almost no effect on the distribution of Z_N. In other words, when D is large enough, the addition or removal of an element from D will cause almost no change to the distribution of Z_N, i.e., differential privacy can be achieved. To illustrate the intuition, we conducted three Monte Carlo simulations. The results are shown in Fig. 2. In each simulation, two sets D_1 and D_2 were used, such that D_1 had N elements and D_2 was obtained from D_1 by adding one extra element. We set $N = 99$, $N = 999$ and $N = 19999$ in the three simulations. Each simulation had 10 million rounds. In each round, we generated a random set of m hash keys, built m sketches for D_1 and m sketches for D_2, and then computed $F(D_i)$ from the sketches. Fig. 2 shows the distributions of $F(D_1)$ and $F(D_2)$, each obtained from 10 million samples. As can be seen, when N becomes larger, the two curves become closer.

Note that our protocol is designed for applications that require one-off or periodical release of statistics (e.g. the number of distinct IP addresses per hour). In each run of our protocol, fresh randomness is introduced by renegotiating the PRF key, so that the sketches are independent of those in the previous run. The protocol does not use sketches from previous runs, and only one query is answered in each run (i.e. the output of each run is Z_N). The protocol does not support correlated queries, e.g. how many new elements have been added since the last estimation. If our protocol is used for answering correlated queries, differential privacy may no longer hold because correlated queries leak more information.

In the following, we will start by showing that when using a single FM sketch, we can find an N_0 such that the protocol satisfies (ε,δ)-differential privacy whenever the input set to the protocol has cardinality at least N_0. Then the bound N_0 for (ε,δ)-differential privacy to hold in the m FM sketches case can be obtained by using the composition theorems of differential privacy [29]. The bound obtained from the composition theorems can be refined, to get a much smaller (better) N_0.

5.2.2 PMF: Single FM Sketch

Let z_N denote the discrete random variable extracted from an FM sketch when the input cardinality is N. We first work out the probability mass function (PMF) of z_N. In [34], the complementary cumulative distribution function of z_N was given as:

$$q_{N,k} = Pr(z_N \geq k) = \sum_{j=0}^{2^k} (-1)^{\nu(j)} e^{-\frac{j\cdot N}{2^k}}$$

where $0 \leq k \leq w-1$ and $\nu(j)$ denotes the number of ones in the binary representation of j. Then, we can derive the PMF of z_N as:

$$\begin{aligned} p_{N,k} &= Pr(z_N = k) = q_{N,k} - q_{N,k+1} \\ &= \sum_{j=0}^{2^k} (-1)^{\nu(j)} e^{-\frac{j\cdot N}{2^k}} - \sum_{i=0}^{2^{k+1}} (-1)^{\nu(i)} e^{-\frac{i\cdot N}{2^{k+1}}} \\ &= \sum_{j=0}^{2^k} (-1)^{\nu(j)} e^{-\frac{j\cdot N}{2^k}} - \left(\sum_{j=0}^{2^k} (-1)^{\nu(2j)} e^{-\frac{2j\cdot N}{2^{k+1}}} \right. \\ &\quad \left. + \sum_{j=0}^{2^k-1} (-1)^{\nu(2j+1)} e^{-\frac{(2j+1)\cdot N}{2^{k+1}}} \right). \end{aligned} \tag{2}$$

In the above, (2) is obtained by summing odd i and even i separately. Then, since the binary representation of $2j$ is almost the same as that of j except one more 0 at the end, we have $\nu(2j) = \nu(j)$. Therefore, the first two terms in (2) are cancelled. Also, because $\nu(2j+1) = \nu(j) + 1$, we obtain

$$p_{N,k} = \sum_{j=0}^{2^k-1} (-1)^{\nu(j)} e^{-\frac{(2j+1)\cdot N}{2^{k+1}}}. \tag{3}$$

Note, the sequence $(-1)^{\nu(j)}$ is actually the Morse-Thue sequence [9]. Let X be an indeterminate, the following always holds for $k \geq 0$ (Proposition 2, [8]):

$$\sum_{j=0}^{2^{k+1}-1} (-1)^{\nu(j)} X^j = \prod_{i=0}^{k} (1 - X^{2^i}). \tag{4}$$

This is because

$$\begin{aligned}\prod_{i=0}^{k}(1-X^{2^i}) =&(1-X^{2^0})(1-X^{2^1})(1-X^{2^2})\cdots(1-X^{2^k})\\ =&1+(-1)^1X^{2^0}+(-1)^1X^{2^1}+(-1)^{1+1}X^{2^1+2^0}+(-1)^1X^{2^2}\\ &+(-1)^{1+1}X^{2^2+2^0}\cdots+(-1)^{k+1}X^{2^k+2^{k-1}+\cdots+2^1+2^0}\\ =&1-X-X^2+X^3-X^4+X^5\cdots+(-1)^{k+1}X^{2^{k+1}-1}\\ =&\sum_{j=0}^{2^{k+1}-1}(-1)^{b(j,0)+\cdots+b(j,k)}X^j\\ =&\sum_{j=0}^{2^{k+1}-1}(-1)^{v(j)}X^j.\end{aligned}$$

In the above, $b(j,l)$, is the lth bit in the binary representation of j (bit numbering starts at 0 for the least significant bit).

We can then rewrite (3), and get the formula for $p_{N,k}$:

$$p_{N,k}=\begin{cases}e^{-\frac{N}{2}} & \text{if } k=0\\ e^{-\frac{N}{2^{k+1}}}\prod_{j=0}^{k-1}(1-e^{-\frac{N}{2^{j+1}}}) & \text{if } k>0\end{cases} \tag{5}$$

The above is obtained as the following: the $k=0$ case is derived directly from (3); when $k>0$, we can let $X=e^{-\frac{N}{2^{k+1}}}$ and use (4):

$$\begin{aligned}p_{N,k}=&\sum_{j=0}^{2^k-1}(-1)^{v(j)}X^{2j+1}=X\left(\sum_{j=0}^{2^k-1}(-1)^{v(j)}X^{2j}\right)\\ =&X\left(\sum_{j=0}^{2^k-1}(-1)^{v(j)}(X^2)^j\right)\\ =&X\prod_{i=0}^{k-1}(1-(X^2)^{2^i})=X\prod_{i=0}^{k-1}(1-X^{2^{i+1}})\\ =&e^{-\frac{N}{2^{k+1}}}\prod_{i=0}^{k-1}(1-e^{-\frac{2^{i+1}N}{2^{k+1}}}).\end{aligned}$$

Let $j=k-i-1$, the final form of the above is:

$$p_{N,k}=e^{-\frac{N}{2^{k+1}}}\prod_{j=0}^{k-1}(1-e^{-\frac{N}{2^{j+1}}}).$$

5.2.3 Finding N_0: Single FM Sketch

We want (ε,δ)-differential privacy to hold for any sufficiently large datasets D_1 and D_2 differing in at most one element. When using a single FM sketch in our protocol, it is equivalent to say that we want to find an N_0 such that for all $N\geq N_0$ and for all k, the following holds:

$$\begin{cases}Pr[z_N=k]\leq e^{\varepsilon}\cdot Pr[z_{N+1}=k]+\delta\\ Pr[z_{N+1}=k]\leq e^{\varepsilon}\cdot Pr[z_N=k]+\delta\end{cases}$$

which is equivalent to:

$$e^{-\varepsilon}\cdot p_{N+1,k}-\frac{\delta}{e^{\varepsilon}}\leq p_{N,k}\leq e^{\varepsilon}\cdot p_{N+1,k}+\delta.$$

It is easy to see that the above holds, if at each k either of the following two conditions is true: (1) $e^{-\varepsilon}\leq\frac{p_{N,k}}{p_{N+1,k}}\leq e^{\varepsilon}$ (ε-differential privacy holds at those k), or (2) $p_{N,k}\leq\delta$ and $p_{N+1,k}\leq\delta$ (the probability of getting to this k is sufficiently small). When condition (1) is true, (ε,δ)-differential privacy holds because

$$e^{-\varepsilon}\cdot p_{N+1,k}-\frac{\delta}{e^{\varepsilon}}<e^{-\varepsilon}\cdot p_{N+1,k}\leq p_{N,k}\leq e^{\varepsilon}\cdot p_{N+1,k}<e^{\varepsilon}\cdot p_{N+1,k}+\delta$$

When condition (2) is true, (ε,δ)-differential privacy holds because

$$e^{-\varepsilon}\cdot p_{N+1,k}-\frac{\delta}{e^{\varepsilon}}<0\leq p_{N,k}\leq\delta\leq e^{\varepsilon}\cdot p_{N+1,k}+\delta$$

To start with, we prove the following lemma:

Lemma 1. *$\frac{p_{N,k}}{p_{N+1,k}}$ decreases monotonically in k.*

Proof. When $k>0$, we have:

$$\frac{p_{N,k}}{p_{N+1,k}}=\frac{e^{-\frac{N}{2^{k+1}}}\prod_{j=0}^{k-1}(1-e^{-\frac{N}{2^{j+1}}})}{e^{-\frac{N+1}{2^{k+1}}}\prod_{j=0}^{k-1}(1-e^{-\frac{N+1}{2^{j+1}}})}=e^{\frac{1}{2^{k+1}}}\cdot\prod_{j=0}^{k-1}\frac{(1-e^{-\frac{N}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})}$$

The first term $e^{\frac{1}{2^{k+1}}}$ decreases monotonically in k. In the product, each term $\frac{1-e^{-\frac{N}{2^{j+1}}}}{1-e^{-\frac{N+1}{2^{j+1}}}}<1$, and the number of terms increases in k. Therefore the value of the product decreases when k increases. So, $\frac{p_{N,k}}{p_{N+1,k}}$ decreases monotonically in k when $k>0$.

When $k=0$, $\frac{p_{N,0}}{p_{N+1,0}}=e^{-\frac{N}{2}+\frac{N+1}{2}}=e^{\frac{1}{2}}$, and it is greater than $\frac{p_{N,1}}{p_{N+1,1}}=e^{\frac{1}{4}}\cdot\frac{1-e^{-\frac{N}{2}}}{1-e^{-\frac{N+1}{2}}}$.

Thus, $\frac{p_{N,k}}{p_{N+1,k}}$ decreases monotonically in k for all $k\geq 0$. $\square$

Looking ahead, based on Lemma 1, our strategy for finding N_0 consists of two steps:

1. Find N_1 such that for all $N\geq N_1$, there exists k_{min}, and (i) for all $k\geq k_{min}$, $\frac{p_{N,k}}{p_{N+1,k}}\leq e^{\varepsilon}$ and (ii) for all $k<k_{min}$, $p_{N,k}\leq\delta$ and $p_{N+1,k}\leq\delta$.
2. Find N_2 such that for all $N\geq N_2$, there exists k_{max}, and (i) for all $k\leq k_{max}$, $\frac{p_{N,k}}{p_{N+1,k}}\geq e^{-\varepsilon}$ and (ii) for all $k>k_{max}$, $p_{N,k}\leq\delta$ and $p_{N+1,k}\leq\delta$.

Then, we take $N_0=max(N_1,N_2)$. Clearly, for all $N\geq N_0$, we have: (i) $e^{-\varepsilon}\leq\frac{p_{N,k}}{p_{N+1,k}}\leq e^{\varepsilon}$ for $k_{min}\leq k\leq k_{max}$, and (ii) $p_{N,k}\leq\delta$ and $p_{N+1,k}\leq\delta$ for $k<k_{min}$ or $k>k_{max}$. Thus,(ε,δ)-differential privacy holds for all $N\geq N_0$ and all k (see Figure 3).

Finding k_{min} and N_1 We first show the existence of k_{min}:

Lemma 2. *Let $k_{min}=max(\lceil\log_2\frac{1}{\varepsilon}\rceil-1,0)$. For any $\varepsilon>0$ and any $k\geq k_{min}$, it holds that $\frac{p_{N,k}}{p_{N+1,k}}\leq e^{\varepsilon}$.*

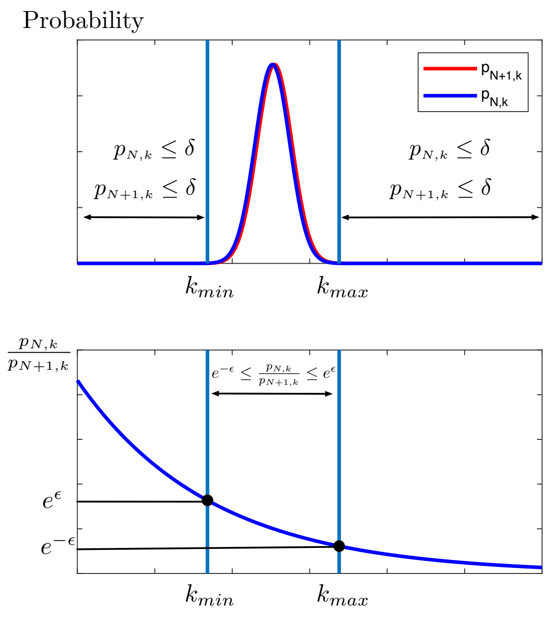


Figure 3: k_{min} and k_{max}

Proof. Recall that for $k=0$, $\frac{p_{N,k}}{p_{N+1,k}} = e^{\frac{1}{2}}$, and for $k>0$, $\frac{p_{N,k}}{p_{N+1,k}} = e^{\frac{1}{2^{k+1}}} \cdot \prod_{j=0}^{k-1} \frac{(1-e^{-\frac{N}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})}$ and $\prod_{j=0}^{k-1} \frac{(1-e^{-\frac{N}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})} < 1$. Therefore, $\frac{p_{N,k}}{p_{N+1,k}} \leq e^{\frac{1}{2^{k+1}}}$ for all $k \geq 0$. It is clear that $\frac{p_{N,k}}{p_{N+1,k}} \leq e^{\varepsilon}$ holds, if $e^{\frac{1}{2^{k+1}}} \leq e^{\varepsilon}$ or equivalently $\frac{1}{2^{k+1}} \leq \varepsilon$. From the last inequality, we get $k_{min} = \lceil \log_2 \frac{1}{\varepsilon} \rceil - 1$. By definition, k is a non-negative integer, so $k_{min} = max(\lceil \log_2 \frac{1}{\varepsilon} \rceil - 1, 0)$. □

Lemma 2 tells us that k_{min} always exists for any $\varepsilon > 0$, and that it is independent of N. Next we will show that the increase of N can eventually make $p_{N,k} \leq \delta$ and $p_{N+1,k} \leq \delta$ for all $k < k_{min}$. First, we show the following lemma:

Lemma 3. *For all $k_{min} > 0$, $Pr[z_N < k_{min}]$ decreases monotonically in N, and $\lim_{N\to\infty} Pr[z_N < k_{min}] = 0$.*

Proof. Using (4), we can rewrite $q_{N,k_{min}}$:

$$\begin{aligned} q_{N,k_{min}} &= Pr(z_N \geq k_{min}) = \sum_{j=0}^{2^{k_{min}}} (-1)^{\nu(j)} e^{-\frac{j \cdot N}{2^{k_{min}}}} \\ &= \sum_{j=0}^{2^{k_{min}}-1} (-1)^{\nu(j)} (e^{-\frac{N}{2^{k_{min}}}})^j - e^{-N} \\ &= \prod_{i=0}^{k_{min}-1} (1-e^{-\frac{N}{2^{k_{min}-i}}}) - e^{-N} = \prod_{j=1}^{k_{min}} (1-e^{-\frac{N}{2^j}}) - e^{-N} \end{aligned}$$

Note that $1-e^{-\frac{N}{2^j}} \geq 0$ and it increases monotonically in N. Thus, $\prod_{j=1}^{k_{min}} (1-e^{-\frac{N}{2^j}})$ increases monotonically in N. Furthermore, $-e^{-N}$ increases monotonically in N. Thus, $q_{N,k_{min}}$ increases monotonically in N. Also, because $Pr[z_N < k_{min}] = 1 - q_{N,k_{min}}$, $Pr[z_N < k_{min}]$ decreases monotonically in N.

Clearly, $\lim_{N\to\infty} q_{N,k_{min}} = 1$, for any $k_{min} > 0$. Thus, $lim_{N\to\infty}(1-q_{N,k_{min}}) = lim_{N\to\infty}(Pr[z_N < k_{min}]) = 0$. □

Now we are ready to state the following theorem:

Theorem 2. *Let $k_{min} = max(\lceil \log_2 \frac{1}{\varepsilon} \rceil - 1, 0)$ and N_1 be the smallest positive integer such that $1 - q_{N_1,k_{min}} \leq \delta$. Then, for all $N \geq N_1$, it holds that $p_{N,k} \leq \delta$ and $p_{N+1,k} \leq \delta$ when $k < k_{min}$, and $\frac{p_{N,k}}{p_{N+1,k}} \leq e^{\varepsilon}$ when $k \geq k_{min}$.*

Proof. When $k_{min} > 0$,

- $1 - q_{N,k_{min}} = Pr[z_N < k_{min}]$ and by Lemma 3, this quantity decreases monotonically in N. Hence if $1 - q_{N_1,k_{min}} \leq \delta$, $\forall N \geq N_1$, $Pr[z_N < k_{min}] \leq \delta$, and because $Pr[z_N < k_{min}] = \sum_{k<k_{min}} p_{N,k}$, $\forall k < k_{min}, p_{N,k} \leq \delta$.
- Because $Pr[z_N < k_{min}]$ decreases monotonically in N, if $Pr[z_N < k_{min}] \leq \delta$ then $Pr[z_{N+1} < k_{min}] \leq Pr[z_N < k_{min}] \leq \delta$, hence $\forall N \geq N_1, \forall k < k_{min}, p_{N+1,k} \leq \delta$.

When $k_{min} = 0$, then for all $k < k_{min}$ it holds that $p_{N,k} = p_{N+1,k} = 0$ because by definition, k cannot be negative. Hence in this case $p_{N,k} < \delta$ and $p_{N+1,k} < \delta$ holds trivially for all $N > 0$. $\frac{p_{N,k}}{p_{N+1,k}} \leq e^{\varepsilon}$ when $k \geq k_{min}$ follows directly from Lemma 2 □

Finding k_{max} and N_2 We now show the existence of k_{max}. Note that unlike k_{min}, k_{max} is a value that is dependant on N.

Lemma 4. *Let $k_{max} = \lceil \log_2 N \rceil + c$ and $c = \lceil \frac{-1+\sqrt{1+8\log_2 \frac{1}{\delta}}}{2} \rceil$. For all $0 < \delta < 1$ and $N \in \mathbb{Z}^+$, $p_{N,k} \leq \delta$ and $p_{N+1,k} \leq \delta$ for all $k > k_{max}$.*

Proof. Let $c \in \mathbb{Z}^+$ and $k > c$, the following holds:

$$p_{N,k} = e^{-\frac{N}{2^{k+1}}} \prod_{j=0}^{k-1} (1-e^{-\frac{N}{2^{j+1}}}) \tag{6}$$

$$< (1-e^{-\frac{N}{2^k}}) \cdots (1-e^{-\frac{N}{2^{k-c}}}) \tag{7}$$

$$< (\frac{N}{2^k}) \cdots (\frac{N}{2^{k-c}}). \tag{8}$$

In the above, (7) holds because it is obtained from (6) by throwing away certain terms whose value is less than 1. (8) holds because by the Maclaurin expansion of e^{-x}, $1-e^{-x} \leq x$ for $0 \leq x \leq 1$.

Then for all $k \geq \lceil \log_2 N \rceil + c$, we have:

$$\begin{aligned} (8) &\leq (\frac{N}{2^{\lceil \log_2 N \rceil + c}}) \cdots (\frac{N}{2^{\lceil \log_2 N \rceil + c - c}}) \\ &\leq (\frac{N}{2^{\log_2 N + c}}) \cdots (\frac{N}{2^{\log_2 N}}) \\ &\leq \frac{1}{2^c} \cdots \frac{1}{2} \cdot 1 = 2^{-(1+2+\cdots+c)}. \end{aligned}$$

To let $p_{N,k} \leq \delta$, it is sufficient to have $2^{-(1+2+\cdots+c)} \leq \delta$, which is equivalent to $\frac{1}{2}c(c+1) \geq \log_2(\frac{1}{\delta})$. Solving this, we get $c \geq \frac{-1+\sqrt{1+8\log_2(\frac{1}{\delta})}}{2}$. Therefore $p_{N,k} \leq \delta$ if $k \geq \lceil \log_2 N \rceil + c$.

Similarly for $p_{N+1,k}$. Also let $c \in \mathbb{Z}^+$ and $k > c$, we have:

$$\begin{aligned} p_{N+1,k} &= e^{-\frac{N+1}{2^{k+1}}}\prod_{j=0}^{k-1}(1-e^{-\frac{N+1}{2^{j+1}}}) \\ &< (1-e^{-\frac{N+1}{2^{k}}})\cdots(1-e^{-\frac{N+1}{2^{k-c}}}) \\ &< (\frac{N+1}{2^k})\cdots(\frac{N+1}{2^{k-c}}) \end{aligned} \tag{9}$$

Then for all $k \geq \lceil \log_2 N \rceil + c + 1$, we have:

$$\begin{aligned} (9) &\leq (\frac{N+1}{2^{\lceil \log_2 N\rceil + c+1}})\cdots(\frac{N+1}{2^{\lceil \log_2 N\rceil + c+1-c}}) \\ &\leq (\frac{N+1}{2^{\lceil \log_2 N\rceil+1}}\cdot\frac{1}{2^c})\cdots(\frac{N+1}{2^{\lceil \log_2 N\rceil+1}}\cdot 1) \\ &\leq \frac{1}{2^c}\cdots\frac{1}{2}\cdot 1 \end{aligned} \tag{10}$$

In the above, (10) holds is because $2^{\lceil \log_2 N\rceil+1} = 2^{\lceil \log_2 N\rceil} + 2^{\lceil \log_2 N\rceil} \geq N+1$, then $\frac{N+1}{2^{\lceil \log_2 N\rceil+1}} \leq 1$. To let $p_{N+1,k} \leq \delta$, it is sufficient to have $2^{-(1+2+\cdots+c)} \leq \delta$, which is equivalent to $\frac{1}{2}c(c+1) \geq \log_2(\frac{1}{\delta})$. Solving this, we get $c \geq \frac{-1+\sqrt{1+8\log_2(\frac{1}{\delta})}}{2}$. Hence $p_{N+1,k} \leq \delta$ if $k \geq \lceil \log_2 N\rceil + c + 1$.

Combine all together, for all $N \in \mathbb{Z}^+$ and all $0 < \delta < 1$, $p_{N,k} \leq \delta$ and $p_{N+1,k} \leq \delta$ when $k \geq \lceil \log_2 N\rceil + \frac{-1+\sqrt{1+8\log_2(\frac{1}{\delta})}}{2} + 1$. We take c to be the smallest integer $c = \lceil \frac{-1+\sqrt{1+8\log_2\frac{1}{\delta}}}{2} \rceil$, then $k > \lceil \log_2 N\rceil + c$ implies $k \geq \lceil \log_2 N\rceil + \frac{-1+\sqrt{1+8\log_2(\frac{1}{\delta})}}{2} + 1$, and hence $p_{N,k} \leq \delta$ and $p_{N+1,k} \leq \delta$. $\square$

Next we want to find N_2 such that for all $N \geq N_2$, $\frac{p_{N,k_{max}}}{p_{N+1,k_{max}}} \geq e^{-\varepsilon}$. If this holds, then by Lemma 1, $\frac{p_{N,k}}{p_{N+1,k}} \geq e^{-\varepsilon}$ for all $k \leq k_{max}$. Recall that for $k = 0$, $\frac{p_{N,k}}{p_{N+1,k}} = e^{\frac{1}{2}} > e^{-\varepsilon}$ for all N trivially. Then we only need to consider the case $k > 0$. In this case, $\frac{p_{N,k}}{p_{N+1,k}} = e^{\frac{1}{2^{k+1}}} \cdot \prod_{j=0}^{k-1}\frac{(1-e^{-\frac{N}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})}$. Let us define

$$\Psi(N,k) = \prod_{j=0}^{k-1}\frac{(1-e^{-\frac{N}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})} \tag{11}$$

We can see if $\Psi(N,k) \geq e^{-\varepsilon}$ then $\frac{p_{N,k}}{p_{N+1,k}} \geq e^{-\varepsilon}$, because $e^{\frac{1}{2^{k+1}}} \geq 1$.

It is actually not difficult to find some N_2 such that $\Psi(N_2, k_{max}) \geq e^{-\varepsilon}$. The tricky part is whether for all $N \geq N_2$, $\Psi(N, k_{max}) \geq e^{-\varepsilon}$ still holds. If $\Psi(N, k_{max})$ is monotonically increasing in N, then this can be proved. However, this is only partially true. Regarding this, we have the following:

Lemma 5. *Let k_{max} as defined in Lemma 4, $\Psi(N,k_{max}) < \Psi(N+1,k_{max})$ if $\lceil \log_2 N\rceil = \lceil \log_2(N+1)\rceil$.*

Proof. Let $\lceil \log_2 N\rceil = \lceil \log_2(N+1)\rceil = t$ and $k_{max} = t+c$,

$$\begin{aligned} \frac{\Psi(N,k_{max})}{\Psi(N+1,k_{max})} &= \frac{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N}{2^{j+1}}})}{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+1}{2^{j+1}}})} \Bigg/ \frac{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+1}{2^{j+1}}})}{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+2}{2^{j+1}}})} \\ &= \frac{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N}{2^{j+1}}})}{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+1}{2^{j+1}}})} \cdot \frac{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+2}{2^{j+1}}})}{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+1}{2^{j+1}}})} \\ &= \prod_{j=0}^{t+c-1}\frac{(1-e^{-\frac{N}{2^{j+1}}})(1-e^{-\frac{N+2}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})^2} \\ &= \prod_{j=0}^{t+c-1}\frac{1-e^{-\frac{N}{2^{j+1}}}-e^{-\frac{N+2}{2^{j+1}}}+e^{-\frac{N}{2^{j+1}}}e^{-\frac{N+2}{2^{j+1}}}}{1-2e^{-\frac{N+1}{2^{j+1}}}+(e^{-\frac{N+1}{2^{j+1}}})^2} \\ &= \prod_{j=0}^{t+c-1}\frac{1-e^{-\frac{N}{2^{j+1}}}-e^{-\frac{N+2}{2^{j+1}}}+e^{-\frac{2N+2}{2^{j+1}}}}{1-2e^{-\frac{N+1}{2^{j+1}}}+e^{-\frac{2N+2}{2^{j+1}}}}. \end{aligned}$$

Observe that:

$$\begin{aligned} & e^{-\frac{N}{2^{j+1}}}+e^{-\frac{N+2}{2^{j+1}}}-2e^{-\frac{N+1}{2^{j+1}}} \\ =& e^{-\frac{N}{2^{j+1}}}+e^{-\frac{N}{2^{j+1}}}e^{-\frac{2}{2^{j+1}}}-2e^{-\frac{N}{2^{j+1}}}e^{-\frac{1}{2^{j+1}}} \\ =& e^{-\frac{N}{2^{j+1}}}(1+e^{-\frac{2}{2^{j+1}}}-2e^{-\frac{1}{2^{j+1}}}) = e^{-\frac{N}{2^{j+1}}}(1-e^{-\frac{1}{2^{j+1}}})^2 \\ >& 0. \end{aligned}$$

Hence, $e^{-\frac{N}{2^{j+1}}}+e^{-\frac{N+2}{2^{j+1}}} > 2e^{-\frac{N+1}{2^{j+1}}}$, and $\frac{\Psi(N,k_{max})}{\Psi(N+1,k_{max})} < 1$, which means $\Psi(N,k_{max}) < \Psi(N+1,k_{max})$. $\square$

In the case that $\lceil \log_2 N\rceil \neq \lceil \log_2(N+1)\rceil$, there is a problem because k_{max} changes. Recall that the value of $k_{max} = \lceil \log_2 N\rceil + c$. In the border case if $N = 2^t - 1$ then $\lceil \log_2 N\rceil = t-1$ and $\lceil \log_2(N+1)\rceil = t$, so we need to compare $\Psi(N, t-1+c)$ and $\Psi(N+1, t+c)$. In this case:

$$\begin{aligned} \frac{\Psi(N,t-1+c)}{\Psi(N+1,t+c)} &= \frac{\prod_{j=0}^{t+c-2}(1-e^{-\frac{N}{2^{j+1}}})}{\prod_{j=0}^{t+c-2}(1-e^{-\frac{N+1}{2^{j+1}}})} \cdot \frac{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+2}{2^{j+1}}})}{\prod_{j=0}^{t+c-1}(1-e^{-\frac{N+1}{2^{j+1}}})} \\ &= \prod_{j=0}^{t+c-2}\frac{(1-e^{-\frac{N}{2^{j+1}}})(1-e^{-\frac{N+2}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})^2} \cdot \left(\frac{1-e^{-\frac{N+2}{2^{t+c}}}}{1-e^{-\frac{N+1}{2^{t+c}}}}\right) \end{aligned} \tag{12}$$

While the product term in (12) is less than 1, the term in the big brackets is greater than 1. It is hard to decide whether the whole formula is less than 1 or not. Although we cannot compare $\Psi(2^t-1, t-1+c)$ and $\Psi(2^t, t+c)$, in Lemma 6 we can show a weaker result (note 2^{t-1} in the lemma instead of 2^t-1):

Lemma 6. *For all $t \in \mathbb{Z}^+$, $\Psi(2^{t-1}, t-1+c) < \Psi(2^t, t+c)$ where c is as defined in Lemma 4.*

Proof.

$$\begin{aligned} \frac{\Psi(2^{t-1},t-1+c)}{\Psi(2^t,t+c)} &= \frac{\prod_{j=0}^{t+c-2}(1-e^{-\frac{2^{t-1}}{2^{j+1}}})}{\prod_{j=0}^{t+c-2}(1-e^{-\frac{2^{t-1}+1}{2^{j+1}}})} \cdot \frac{\prod_{j=0}^{t+c-1}(1-e^{-\frac{2^t+1}{2^{j+1}}})}{\prod_{j=0}^{t+c-1}(1-e^{-\frac{2^t}{2^{j+1}}})} \\ &= \left(\prod_{j=0}^{t+c-2}\frac{(1-e^{-\frac{2^{t-1}}{2^{j+1}}})(1-e^{-\frac{2^t+1}{2^{j+1}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{j+1}}})(1-e^{-\frac{2^t}{2^{j+1}}})}\right) \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c}}})}{(1-e^{-\frac{2^t}{2^{t+c}}})}\right). \end{aligned} \tag{13}$$

Let us keep only the last two terms in the product (when $j = t+c-3$ and $j = t+c-2$), we have:

$$
\begin{aligned}
(13) < {} & \left(\frac{(1-e^{-\frac{2^{t-1}}{2^{t+c-2}}})(1-e^{-\frac{2^t+1}{2^{t+c-2}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-2}}})(1-e^{-\frac{2^t}{2^{t+c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{2^{t-1}}{2^{t+c-1}}})(1-e^{-\frac{2^t+1}{2^{t+c-1}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-1}}})(1-e^{-\frac{2^t}{2^{t+c-1}}})} \right) \\
& \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c}}})}{(1-e^{-\frac{2^t}{2^{t+c}}})} \right) \\
= {} & \left(\frac{(1-e^{-\frac{1}{2^{c-1}}})(1-e^{-\frac{2^t+1}{2^{t+c-2}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-2}}})(1-e^{-\frac{1}{2^{c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{1}{2^c}})(1-e^{-\frac{2^t+1}{2^{t+c-1}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-1}}})(1-e^{-\frac{1}{2^{c-1}}})} \right) \\
& \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c}}})}{(1-e^{-\frac{1}{2^c}})} \right) \\
= {} & \left(\frac{\cancel{(1-e^{-\frac{1}{2^{c-1}}})}(1-e^{-\frac{2^t+1}{2^{t+c-2}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-2}}})(1-e^{-\frac{1}{2^{c-2}}})} \right) \cdot \left(\frac{\cancel{(1-e^{-\frac{1}{2^c}})}(1-e^{-\frac{2^t+1}{2^{t+c-1}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-1}}})\cancel{(1-e^{-\frac{1}{2^{c-1}}})}} \right) \\
& \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c}}})}{\cancel{(1-e^{-\frac{1}{2^c}})}} \right) \\
= {} & \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c-2}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-2}}})(1-e^{-\frac{1}{2^{c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c-1}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-1}}})} \right) \cdot (1-e^{-\frac{2^t+1}{2^{t+c}}}) \\
= {} & \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c-2}}})}{(1-e^{-\frac{1}{2^{c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c-1}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{2^t+1}{2^{t+c}}})}{(1-e^{-\frac{2^{t-1}+1}{2^{t+c-1}}})} \right) \\
= {} & \left(\frac{(1-e^{-\frac{1+2^{-t}}{2^{c-2}}})}{(1-e^{-\frac{1}{2^{c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{2^{-1}+2^{-t-1}}{2^{c-2}}})}{(1-e^{-\frac{2^{-1}+2^{-t}}{2^{c-2}}})} \right) \cdot \left(\frac{(1-e^{-\frac{2^{-2}+2^{-t-2}}{2^{c-2}}})}{(1-e^{-\frac{2^{-2}+2^{-t-1}}{2^{c-2}}})} \right).
\end{aligned}
$$

Then let $b = e^{-2^{-c}}$ and $y = e^{-2^{-t-c}}$, the above can be rewritten as:

$$
\begin{aligned}
& \frac{1-b^4y^4}{1-b^4} \cdot \frac{1-b^2y^2}{1-b^2y^4} \cdot \frac{1-by}{1-by^2} \\
= & \frac{1-by-b^2y^2+b^3y^3-b^4y^4+b^5y^5+b^6y^6-b^7y^7}{1-b^4-(b-b^5)y^2-(b^2-b^6)y^4+(b^3-b^7)y^6} = \frac{f_1(y)}{f_2(y)}.
\end{aligned}
$$

Let $f(y) = f_1(y) - f_2(y)$, we can prove that when $b \in (e^{-1}, 1)$, $f(y) < 0$ for all $y \in (b, 1)$ (see appendix H). This means for any $c \in (0, +\infty)$ and $t \in \mathbb{Z}^+$, $\Psi(2^{t-1}, t-1+c) < \Psi(2^t, t+c)$, and that the lemma statement is true. □

Lemma 6 is useful because combining it and Lemma 5, we can prove the following lemma:

Lemma 7. *Let $t_0 \in \mathbb{Z}^+$, if $\Psi(2^{t_0}, t_0+c) \geq e^{-\varepsilon}$, then for any $N \geq 2^{t_0}, \varepsilon > 0$, $\Psi(N, k_{max}) \geq e^{-\varepsilon}$, where k_{max} is as defined in Lemma 4.*

Proof. For all $N \geq 2^{t_0}$, we can write $N = 2^{t_0+i} + j$ for some $i \geq 0$ and $0 \leq j \leq 2^{t_0+i} - 1$. When $j = 0$, by Lemma 6, we can see that $\Psi(N, k_{max}) = \Psi(2^{t_0+i}, t_0+i+c) > \Psi(2^{t_0+i-1}, t_0+i-1+c) > \cdots > \Psi(2^{t_0}, t_0+c)$. When $j \neq 0$, by Lemma 5, we can see that $\Psi(N, k_{max}) = \Psi(2^{t_0+i}+j, t_0+i+c) > \Psi(2^{t_0+i}, t_0+i+c)$, then by Lemma 6 $\Psi(2^{t_0+i}, t_0+i+c) > \Psi(2^{t_0}, t_0+c)$. Therefore $\Psi(N, k_{max}) \geq \Psi(2^{t_0}, t_0+c) \geq e^{-\varepsilon}$. □

Now we are ready to state the next theorem:

Theorem 3. *Let $\varepsilon > 0$, and c, k_{max} as defined in Lemma 4. Let t_0 be the smallest positive integer that satisfies $\Psi(2^{t_0}, t_0+c) \geq e^{-\varepsilon}$. Let N_2 be the smallest integer in $(2^{t_0-1}, 2^{t_0}]$ such that $\Psi(N_2, t_0+c) \geq e^{-\varepsilon}$. Then, (1) $\forall N \geq N_2, k \leq k_{max}$, $\frac{p_{N,k}}{p_{N+1,k}} \geq e^{-\varepsilon}$, and (2) $\forall N \geq N_2, k > k_{max}$, $p_{N,k} \leq \delta$ and $p_{N+1,k} \leq \delta$.*

Proof. We first show (1): for all $N \geq N_2$, there are two cases: (i) when $N \in [N_2, 2^{t_0}]$, then by the definition of N_2, $\Psi(N, k_{max}) \geq e^{-\varepsilon}$, which implies $\frac{p_{N,k_{max}}}{p_{N+1,k_{max}}} \geq e^{-\varepsilon}$. (ii) when $N > 2^{t_0}$, by Lemma 7 $\Psi(N, k_{max}) \geq e^{-\varepsilon}$, which implies $\frac{p_{N,k_{max}}}{p_{N+1,k_{max}}} \geq e^{-\varepsilon}$. Then by Lemma 1, if $\frac{p_{N,k_{max}}}{p_{N+1,k_{max}}} \geq e^{-\varepsilon}$, then for all $k \leq k_{max}$ we have $\frac{p_{N,k}}{p_{N+1,k}} \geq e^{-\varepsilon}$, hence complete the proof of the first part.

(2) follows directly from Lemma 4. □

Computing N_0 Combining all the above together, we can use Algorithm 8 to compute N_0 for a given (ε, δ) pair:

Algorithm 8: *FindN*$_0(\varepsilon, \delta)$

```
Input: $\varepsilon > 0, 0 < \delta < 1$
Result: $N_0 \in \mathbb{Z}^+$
$k_{min} = max(\lceil \log_2 \frac{1}{\varepsilon} \rceil - 1, 0)$;
/* $1 - q_{N,k_{min}}$ decreases monotonically in $N$.     */
Starting from 1, use an exponential search in $[1, +\infty]$ to find
  $N_1$ that is the smallest integer satisfying
  $1 - q_{N_1,k_{min}} = 1 - \sum_{j=0}^{2^{k_{min}}} (-1)^{\nu(j)} e^{-\frac{j \cdot N_1}{2^{k_{min}}}} \leq \delta$;
$c = \lceil \frac{-1+\sqrt{1+8\log_2 \frac{1}{\delta}}}{2} \rceil$;
Starting from 1, use an exponential search in $[1, +\infty]$ to find
  $t_0$ that is the smallest integer satisfying
  $\prod_{j=0}^{k_{max}-1} \frac{(1-e^{-\frac{2^{t_0}}{2^{j+1}}})}{(1-e^{-\frac{2^{t_0}+1}{2^{j+1}}})} \geq e^{-\varepsilon}$, where $k_{max} = t_0 + c$;
/* search backwardly in $(2^{t_0-1}, 2^{t_0}]$     */
for $i = 2^{t_0}; i > 2^{t_0-1}; i--$ do
    if $(\prod_{j=0}^{t_0+c-1} \frac{(1-e^{-\frac{N}{2^{j+1}}})}{(1-e^{-\frac{N+1}{2^{j+1}}})} < e^{-\varepsilon})$ then
        $N_2 = i + 1$;
        break;
    end
end
Output $N_0 = max(N_1, N_2)$;
```

Regarding the algorithm, we have the following theorem:

Theorem 4. *For all $\varepsilon, \delta \in \mathbb{R}^+$ and $\delta \in (0, 1)$, let $N_0 = FindN_0(\varepsilon, \delta)$. When all DPs use a single FM sketch, our protocol satisfies (ε, δ)-differential privacy if the cardinality of the union of all DP's set is greater or equal to N_0.*

The proof is straightforward, given the already proven theorems, and therefore it is omitted.

The running time of Algorithm 8 is bounded by the search time, and in turn the values of, N_1 and N_2. We have the following Theorem:

Theorem 5. *In algorithm 8, N_1 and N_2 increase monotonically as the parameter ε or δ decrease.*

Proof. First we show that as ε decreases, the value of N_1 increases monotonically. When ε decreases, $\frac{1}{\varepsilon}$ increases monotonically, then $k_{min} = max(\lceil \log_2 \frac{1}{\varepsilon} \rceil - 1, 0)$ increases monotonically (weakly). We know that $q_{N_1,k_{min}} = \sum_{k<k_{min}} Pr[z_{N_1} = k]$ decreases monotonically in k_{min}, then $1 - q_{N_1,k_{min}}$ increases monotonically in k_{min}. Therefore with a smaller ε, we gets a larger $1 - q_{N_1,k_{min}}$. We know also that $q_{N_1,k_{min}}$ increases monotonically in N_1. So if we want $1 - q_{N_1,k_{min}} \geq \delta$ to hold, we need to increase N_1.

Next we show that as δ decreases, the value of N_1 increases monotonically. This is obvious, because with a smaller δ, if we want $1 - q_{N_1,k_{min}} \geq \delta$ to hold, $q_{N_1,k_{min}}$, and in turn N_1, must increase.

Then we show that as ε decreases, the value of N_2 increases monotonically. When ε decreases, $e^{-\varepsilon}$ increases monotonically. Then by Lemma 5 and 6, the smallest N_2 to ensure $\prod_{j=0}^{t_0+c-1} \frac{(1-e^{-\frac{N_2}{2^{j+1}}})}{(1-e^{-\frac{N_2+1}{2^{j+1}}})} \geq e^{-\varepsilon}$ must increase monotonically.

Last we show that as δ decreases, the value of N_2 increases monotonically (weakly). When δ decreases, $\frac{1}{\delta}$ increases monotonically, and there are two cases:

- When δ decreases, $c = \lceil \frac{-1+\sqrt{1+8\log_2 \frac{1}{\delta}}}{2} \rceil$ and $k_{max} = t_0 + c$ remains unchanged, hence $\prod_{j=0}^{k_{max}-1} \frac{(1-e^{-\frac{2^{t_0}}{2^{j+1}}})}{(1-e^{-\frac{2^{t_0}+1}{2^{j+1}}})}$ remains unchanged. In this case, N_2 remains unchanged.
- When δ decreases, $c = \lceil \frac{-1+\sqrt{1+8\log_2 \frac{1}{\delta}}}{2} \rceil$ and $k_{max} = t_0 + c$ increase, and in turn the number of terms in $\prod_{j=0}^{k_{max}-1} \frac{(1-e^{-\frac{2^{t_0}}{2^{j+1}}})}{(1-e^{-\frac{2^{t_0}+1}{2^{j+1}}})}$ increases. As the value of every term in it is less than 1, then the value of it, as the product, decreases monotonically. To ensure the value is greater than or equal to $e^{-\varepsilon}$, by Lemma 6, we know the value of t_0 should increase. Hence the value of N_2 increases because $t_0 = \log_2(N_2)$.

$\square$

Therefore for smaller (ε, δ), the algorithm will take longer to run. However this will not be a problem in practice. As an example, we ran Algorithm 8 with extremely small parameters $\varepsilon = 2^{-40}$ and $\delta = 2^{-80}$, $N_0 = max(N_1, N_2)$ found by the algorithm is 30,865,997,083,798, and the running time was in the order of seconds[3]. Therefore, for all (ε, δ) normally used in practice, N_1, N_2 will not be too large and the algorithm can be efficiently computed (see also Table 1, in which the values were computed with $\frac{\varepsilon}{m}, \frac{\delta}{m}$).

[3]The implementation is based on Arb (`http://arblib.org/`), a C library supporting arbitrary precision real arithmetic.

ε \ m	100	1000	2000	4000
1	2053	4596	9387	9564
0.5	4123	9210	18791	19146
0.3	8261	18437	37601	38310
0.2	8261	36891	37601	76638
0.1	16538	73800	75219	153295

Table 1: The value of $\widehat{N_0}$ for different ε, m and fixed $\delta = 2^{-40}, w = 32$

5.2.4 Find N_0: Multiple Sketches

The Bound By Composition Theorems If the DPs use m FM sketches, then the output of the protocols is $Z_N = z_{N,1} + \cdots + z_{N,m}$, where $z_{N,i}$ is extracted from the i-th FM sketch. The input set encoded by each FM sketch is the same, i.e. the union of observations from all DPs, and the hash keys are different. Therefore, using m FM sketches is like querying a privacy mechanism m times, and the randomization of the mechanism is independent for each query. The basic composition theorem (Theorem 3.16, [29]) states that if the base differential privacy mechanism is $(\varepsilon_0, \delta_0)$-differentially private, then after m queries, any function of the m query results is at least $(m\varepsilon_0, m\delta_0)$-differentially private. Therefore, in the m sketches case, given the target (ε, δ) we want to achieve, it suffices if each single FM sketch satisfies $(\frac{\varepsilon}{m}, \frac{\delta}{m})$-differential privacy. When m is large, the advanced composition theorem (Theorem 3.20, [29]) gives a better bound. For a base mechanism that is $(\varepsilon_0, \delta_0)$ differentially private, after m queries, the result is at least $(\varepsilon, m\delta_0 + \delta')$-differential privacy, where

$$\varepsilon = \sqrt{2m \ln \frac{1}{\delta'}}\varepsilon_0 + m\varepsilon_0(e^{\varepsilon_0} - 1), \text{for any } \delta' > 0. \quad (14)$$

Hence, given the target (ε, δ), we can obtain $(\varepsilon_0, \delta_0)$, then an initial bound $\widehat{N_0} = findN_0(\varepsilon_0, \delta_0)$. For all $N \geq \widehat{N_0}$, (ε, δ)-differential privacy holds, due to Theorem 4 and the composition theorems.

In Table 1, we show some $\widehat{N_0}$ for different combinations of parameters. When $m = 100$, the basic composition theorem gives better results, so we set $\varepsilon_0 = \frac{\varepsilon}{100}, \delta_0 = \frac{\delta}{100}$. For all other m, we obtain ε_0, δ_0 through the advanced composition theorem. We simply set $\delta' = \frac{\delta}{2}$ and $\delta_0 = \frac{\delta}{2m}$, then we can get ε_0 by (14). Note that in the table, when $m = 100$, we get the same $\widehat{N_0}$ in the cases when $\varepsilon = 0.2$ and $\varepsilon = 0.3$. This is because in both cases $N_1 > N_2$, so $\widehat{N_0} = N_1$. The value of N_1 is a function of $\lceil \log_2 \frac{1}{\varepsilon_0} \rceil - 1$ and δ_0. The same δ_0 is used in both cases and $\lceil \log_2 \frac{100}{0.2} \rceil - 1 = \lceil \log_2 \frac{100}{0.3} \rceil - 1$, so the algorithm gives the same $\widehat{N_0}$. For the same reason, we get the same $\widehat{N_0}$ for $\varepsilon = 0.2$ and $\varepsilon = 0.3$ when $m = 2000$.

The bound $\widehat{N_0}$ by composition theorems is rather loose and can be further improved. Next we will first show how to compute the PMF of Z_N, then how we can get an improved bound N_0 computationally.

PMF: *m* FM sketches The PMF of Z_N can be obtained through the probability generating functions (pgf for short) [36]. We know that the pgf of a discrete random variable X taking values in non-negative integer $[0, j]$ is defined as:

$$G_X(t) = \mathbf{E}(t^X) = \sum_{k=0}^{j} Pr[X = k] \cdot t^k.$$

Therefore for $z_{N,i}$, the pgfs are:

$$G_{z_{N,i}}(t) = \sum_{k=0}^{w-1} p_{N,k} \cdot t^k.$$

We use pgfs here because they are particularly useful for dealing with the sum of independent random variables. In fact, for $Z_N = \sum_{i=1}^{m} z_{N,i}$, the pgf is:

$$G_{Z_N}(t) = \left(G_{z_{N,i}}(t)\right)^m = \left(\sum_{k=0}^{w-1} p_{N,k} \cdot t^k\right)^m. \tag{15}$$

Another property of a pgf is that the PMF of X can be recovered by taking derivatives of $G_X(t)$:

$$Pr[X = k] = \frac{G_X^{(k)}(0)}{k!}. \tag{16}$$

Expanding $G_{Z_N}(t)$, we will get the $m(w-1)$-th degree polynomials $\sum_{K=0}^{m(w-1)} a_K t^K$, where a_K are coefficients and t is the indeterminate. Then by (16), we have:

$$Pr[Z_N = K] = \frac{G_{Z_N}^{(K)}(0)}{K!} = a_K. \tag{17}$$

Refining the Bound In the m FM sketches case, (ε, δ)-differential privacy holds if for every $0 \le K \le m(w-1)$:

$$e^{-\varepsilon} \cdot Pr[Z_{N+1} = K] - \frac{\delta}{e^{\varepsilon}} \le Pr[Z_N = K] \le e^{\varepsilon} \cdot Pr[Z_{N+1} = K] + \delta. \tag{18}$$

Therefore, we can use algorithm 9 to find the improved N_0.

Algorithm 9 starts from $\widehat{N_0}$ and computationally verifies $N < \widehat{N_0}$ backwardly. It stops at N_0 when $N_0 - 1$ does not satisfy differential privacy anymore. This N_0 is the improved bound and it is guaranteed that for all $N \ge N_0$, our protocol satisfies (ε, δ)-differential privacy at the given (m, w) parameters. In Table 2, we show the improved bound computed from Algorithm 9. Compared to the values in Table 1, the improved bound is significantly better.

The running time of Algorithm 9 is dominated by Step 5, in which the pgfs are computed. Computing pgfs involving polynomial exponentiation and the time increases when m increases. For example, to get numbers in Table 2, it took 78 ms, 5350 ms, 21468 ms and 90237 ms to compute a single $G_{Z_{N_0}}(t)$ when $m = 100, 1000, 2000, 4000$ respectively. When $\widehat{N_0}$ is large, backward verification by Algorithm 9 could take quite long time. That said, it should be noted that this verification needs only to be done once for each parameter combination.

Algorithm 9: $RefineBound(\varepsilon, \delta, \widehat{N_0}, m, w)$

```
   Input: $\varepsilon, \delta \in \mathbb{R}^+$ and $\delta \in (0,1), \widehat{N_0}, m, w \in \mathbb{Z}^+$
   Result: $N_0 \in \mathbb{Z}^+$
1  stop =false;
2  $N_0 = \widehat{N_0}$+1;
3  do
4      $N_0 = N_0 - 1$;
5      Compute the polynomials $G_{Z_{N_0}}(t)$ and $G_{Z_{N_0-1}}(t)$ using (15);
6      for $K = 0; K \le m(w-1); K{+}{+}$ do
7          Let $Pr[Z_N = K]$ be the coefficient of the $K$-th degree term
             of $G_{Z_{N_0-1}}(t)$;
8          Let $Pr[Z_{N+1} = K]$ be the coefficient of the $K$-th degree
             term of $G_{Z_{N_0}}(t)$;
9          if $Pr[Z_N = K]$ and $Pr[Z_{N+1} = K]$ don't satisfy (18) then
10             stop =true;
11             break;
12         end
13     end
14 while stop = false and $N_0 > 0$;
15 output $N_0$
```

ε \ m	100	1000	2000	4000
1	85	254	355	497
0.5	166	496	693	969
0.3	273	813	1136	1587
0.2	404	1205	1682	2351
0.1	790	2359	3293	4600

Table 2: The value of N_0 by Algorithm 9 for different ε, m and fixed $\delta = 2^{-40}, w = 32$

The bound N_0 can easily be achieved in real world applications. For example, when $\varepsilon = 0.3$, which is recommended for safe measurements in anonymity networks [41], even with a large $m = 4000$, N_0 is only 1587. For smaller ε values, N_0 are still reasonably small across different m values. Note that N_0 is the lower bound, therefore the privacy level is guaranteed even if the actual cardinality is larger than N_0. We can also see that for the same privacy parameters, a larger set allows us to get a better accuracy (by allowing a larger m at the same privacy level). This means we can get both good utility and good privacy if the set is large.

6 Experimental Evaluation

We have implemented a prototype of our protocol in C++. The source code of the protocol is available online[4]. We used the implementation of Overdrive (low gear) in the SPDZ2 repository[5] for the pre-processing part, and implemented our offline and online protocols on top of that. We compare the performance of our protocol to the state of the art [33]. The

[4]https://github.com/saftoes/pdce
[5]https://github.com/bristolcrypto/SPDZ-2

			$N = 20000$			$N = 10^6$			$N = 10^9$		
			m=1000	m=2000	m=4000	m=1000	m=2000	m=4000	m=1000	m=2000	m=4000
Running Time (s)	LAN	Offline	66.5	132.2	222.8	78.1	154.7	307.6	149.3	257.3	515.8
		Online	0.079	0.151	1.997	0.110	0.189	0.271	0.201	0.377	0.522
	WAN	Offline	320	624.1	1470.5	411.7	811.1	1578.9	757.1	1421.8	2944.2
		Online	2.414	2.036	2.623	1.754	2.360	2.934	2.689	3.031	5.026
Communication (GB)	Offline		10.7	21.4	35.2	12.09	24.18	48.5	23.3	39.05	78.3
	Online		0.008	0.016	0.031	0.010	0.020	0.041	0.028	0.056	0.120

Table 3: Total running time and communication cost: 5 CPs (16 threads), 20 DPs.

implementation of [33] provided by the authors is in Go and does not fully support multi-threading. For a fair comparison, we re-implemented the protocol in [33] in C++. In this implementation, we use OpenSSL 1.0.1 for all cryptographic operations and pthread for multi-threading. The performance of our new implementation is much better than that reported in [33]. We used 40 for the statistical security parameter and 128 for the computational security parameter in all experiments.

We ran all CPs in Amazon AWS. We used the EC2 instance type r5.4xlarge (on-demand) for each CP. Each instance has 16 vCPUs (8 physical cores) based on Intel Xeon Platinum 8000 series (Skylake-SP) CPUs, 128GB RAM, one network interface up to 10 Gpbs LAN speed, and costs \$1.008 - \$1.12 per hour in US data centers. We conducted experiments both in a LAN environment (all CPs were in the Oregon AWS data center), and a WAN environment (CPs were distributed in 4 different AWS data centers in the US[6]). The DPs ran on desktops, with a typical hardware configuration of an Intel Quadcore i7-6700k CPU and 16 GB RAM. We used 20 DPs in all experiments and varied the number of CPs.

In Table 3, we show the total running time and communication (send+receive) cost of our protocol in the offline and online phases. We implemented the group authenticated key exchange protocol in [42]. The offline phase measurement includes the costs of the SPDZ pre-processing protocol and our offline protocols. The online phase measurement includes all online protocols, from DP sharing the sketch to the CPs outputting Z_N (using the ExtractZ protocol in LAN and ExtractZBS protocol in WAN). Note we do not include the time used by the DPs to collect data because this time is irrelevant to our protocol. In the experiments, the DPs first did the initial sharing and then immediately the final sharing of the Oblivious FM sketches. The running time and communication cost shown in the table are the average of those measured over all CPs. For the running time, we show the time measured in LAN and WAN. The communication costs in LAN and WAN are almost the same, thus we only show the larger one of the two. We varied the number of distinct elements in the experiments, from 20000, to 1 million (10^6), to 1 billion (10^9). This change affects the size of the modulus p (55, 60, 70 respectively) and the size of the sketches w (19, 24, 34 respectively). We also used different number of sketches (m) for different accuracy levels. As we can see in the table, the total running time is dominated by the offline phase. While the offline running time is in the order of minutes, the online running time is only in the order of seconds. We can also see that the offline running time is less than 1 hour even with the largest parameter group, and since the offline computation can be done during the period when the DPs are collecting the data, the performance should be acceptable (many applications may only require daily or even less frequent update of the estimate). The protocol has good scalability: when N increases from 20000 to 10^9 (50000 times), the running time increases only to about 2 times ($\log(10^9)/\log(20000) \approx 2$). The running time in LAN is much less than that measured in WAN. The differences in network bandwidth and latency are likely the causes of the slowdown. Communication-wise, the offline phase cost is much higher than the online phase cost. As we can see in Table 4, most of the cost in the offline phase is due to the Triple generation protocol in SPDZ, which utilizes heavy machinery such as somewhat homomorphic encryption and zero-knowledge proofs. In Table 4, we also show the differences in performance for the ExtractZ (Protocol 7) and ExtractZBS (Protocol 13, Appendix E). The results confirm that in the high network latency setting, ExtractZBS performs better due to fewer communication rounds/interactions.

		Running Time (s)		Comm. (GB)
		LAN	WAN	
Group AKE (per DP)		0.014	0.46	6.36×10^{-6}
Offline	Triple	417.0	2414.1	68.5
	Rand	50.6	452.4	7.4
	Rand2	47.4	70.3	1.83
	RandExp	0.8	7.4	0.61
Online	Share (per DP)	0.155	1.877	0.0087
	MergeShare	0.00130	0.00129	N/A
	ZeroTest	0.32	2.345	0.070
	ExtractZ	0.049	1.482	0.034
	ExtractZBS	0.063	0.803	0.042

Table 4: Performance breakdown: 5 CPs (16 threads), 20 DPs, $N = 10^9$, $m = 4000$

As a comparison, we show in Figure 4 the total running time and communication cost of the protocol in [33]. In the experiments, we used 5 CPs (16 threads) and 20 DPs. We varied N from 20000 to 50000, and as in [33], set the number of bins to $10 \cdot N$ so the collision probability is less than 10%. We also set (ε, δ) for differential privacy to $(0.3, 10^{-12})$, the

[6]N Virginia, Ohio, Northern California, Oregon.

default values used in [33]. Note the parameters are weaker than those for our protocol: with $N = 20000$ and other parameters in the experiments, our protocol can easily achieve $(0.1, 10^{-12})$-differential privacy, and even better privacy when N grows bigger. We only tested with all CPs in the same LAN, as the figures in the WAN setting would be even higher. As we can see, the protocol in [33] is much slower than ours, and its running time increases much faster. When $N = 20000$, its running time in LAN is about 1.2 times of ours in WAN, and 8.5 times of ours in LAN (both $m = 4000$); when $N = 50000$, it needs almost 2.5 hours in LAN, while our protocol (in WAN) with $N = 10^9$ only needs less than 50 minutes (offline+online). The running time of [33] is slightly convex due to a quadratic step in a zero-knowledge proof sub-protocol. The communication complexity of the protocol in [33] is linear. When N is small, the protocol in [33] has a much smaller communication cost compared to ours, e.g. 1.4 GB vs 35.2 GB when $N = 20000$. However since the communication complexity of the protocol in [33] is linear and that of ours is logarithmic, the communication cost of the protocol in [33] will exceed that of ours eventually. As an estimation, when N is 10^6, the communication cost of the protocol in [33] would be 60 GB roughly, which is already higher than ours (48.5 GB).

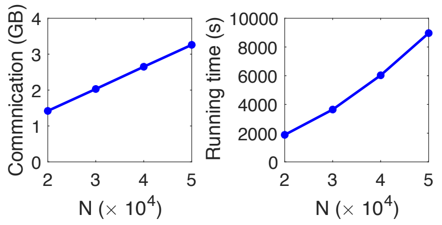
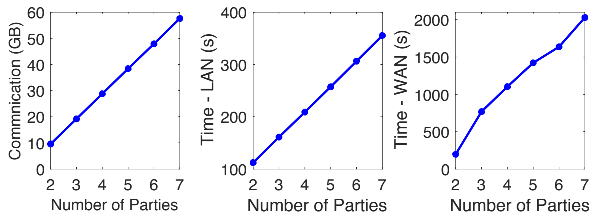


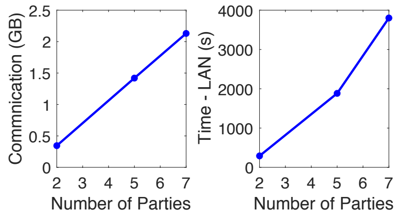
Figure 4: Performance of protocol in [33] (LAN)

Next, we show in Figure 5 the performance of our protocol and the protocol in [33] with a varying number of CPs. For our protocol, we fixed N to 10^9 and m to 4000, with a varying number of CPs from 2 to 7. As we can see, the communication cost and the running time in LAN increase linearly in the number of CPs. The line of the running time in WAN is not very regular, but we can see that the running time is roughly linear. In typical applications, the number of CPs is quite unlikely to exceed 10. However in the case of more CPs, we could switch the SPDZ pre-processing protocol to High Gear. High Gear's performance surpasses Low Gear (we currently use) when executed with a high number of parties (more than 10 as reported in [44]). As the computation time of our protocol is dominated by the SPDZ pre-processing, this would allow us to handle more CPs more gracefully. For the protocol in [33], we fixed N to 20000, and used 2, 5, 7 CPs in the experiment. The communication cost of this protocol is also linear in the number of CPs, but the running time is slightly worse than linear. The results are consistent with those reported in [33].

In Figure 6, we show the distribution of the relative errors ($\frac{|\tilde{N}-N|}{N}$ where $\tilde{N}$ is the cardinality estimated from the sketches



(a) Our protocol, $N = 10^9$, $m = 4000$



(b) Protocol in [33], $N = 20000$

Figure 5: Performance with different number of CPs

and N is the true cardinality) when using a different number of FM sketches. We used $m = 1000$, 2000 and 4000 sketches, using two sets with 20000 and 10^6 random elements as inputs. We repeated each experiment 1000 times and drew the histograms. As we can see, when m increases, the max relative error decreases, and the distribution gets more concentrated towards 0. With $m = 4000$, about 99% of the estimations have a relative error less than 3%, and the maximum relative error observed was 4.3%. On the other hand, the estimations using the method in [33] had a slightly higher relative error (see Figure 10 in Appendix G) due to the hash collisions and the noise added to achieve differential privacy.

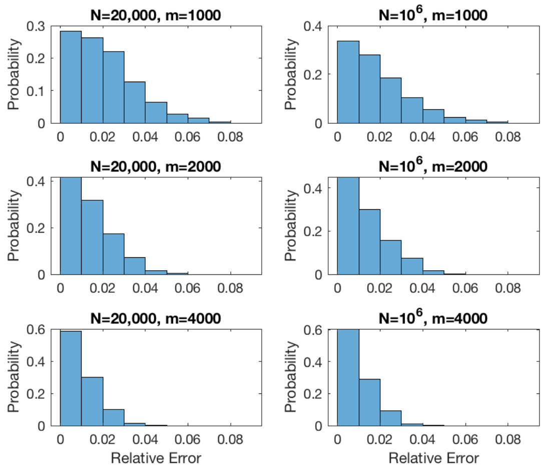


Figure 6: Distribution of relative errors

Since the cardinality count produced by the protocol in [33] is also approximate, it would be interesting to see whether our differential privacy analysis can result in a cheaper variant of that protocol, and if so how would the performance of the variant compare to that of our protocol. In principle the protocol of [33] could also obtain differential privacy for free with honest DPs and a private hash key, although we have not done the analysis and the analysis may not be trivial. If [33] achieves differential privacy by hashing, then the CPs do not need to add noise. However, the performance improvement would be around 20-30% at most, based on our experience of implementing the protocol. The performance would be in the same order as it is now, and thus still much worse than that of our protocol. This is because the main factors affecting the performance of [33] are not adding noise but (1) public key encryption; (2) verifiable shuffling and zero-knowledge proofs; (3) superlinear (in the maximum measurable cardinality) computational and communication complexity.

In Appendix G, we show additional experimental results, which could not be presented here due to limited space.

7 Conclusion and Future Work

In this paper, we present and analyse a PDCE protocol. The protocol is efficient and scalable, due to the use of FM sketches as the underlying data structure for cardinality estimation, and the use of efficient secret sharing based MPC primitives. We proved the security of the protocol against a malicious adversary in the UC framework. More interestingly, we showed that the combination of secure computation and the FM sketches allows us to get (ε, δ)-differential privacy for free. We implemented our protocol and evaluated it experimentally. Our experiments showed that the protocol is much more efficient and scalable than the state of the art [33].

We would like to continue investigating the use of data structures in secure computation protocols to improve their efficiency and scalability. Data structures such as sketches could lead to sub-linear complexity protocols, which are highly desirable for Big Data applications. We would also like to investigate the relationship between differential privacy and sketches, to extend and generalize the results in this paper to other sketches/data structures.

Acknowledgement

We thank shepherd Mathias Lécuyer as well as the anonymous reviewers for their insightful comments. This research was supported in part by UK EPSRC under grant EP/M013561/2; National Natural Science Foundation of China under grant 61722203 (Outstanding Youth Foundation), U1936218 (Joint Fund Project), 62032012 and 62072132; National Key Research and Development Program of China under grant 2020YFB1005700.

References

[1] Health Insurance Portability and Accountability Act of 1996. https://aspe.hhs.gov/report/health-insurance-portability-and-accountability-act-1996, 1996.

[2] Gramm-Leach-Bliley Act. https://www.ftc.gov/tips-advice/business-center/privacy-and-security/gramm-leach-bliley-act, 1999.

[3] General Data Protection Regulation. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN, 2018.

[4] The Royal Society Report on Privacy Enhancing Technologies. https://royalsociety.org/-/media/policy/projects/privacy-enhancing-technologies/privacy-enhancing-technologies-report.pdf, 2019.

[5] Gergely Ács and Claude Castelluccia. A case study: privacy preserving release of spatio-temporal density in paris. In *KDD*, pages 1679–1688, 2014.

[6] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

[7] Akamai. Real-Time Web Metrics Methodology. https://www.akamai.com/uk/en/resources/visualizing-akamai/real-time-web-monitor/real-time-web-metrics-methodology.jsp.

[8] Jean-Paul Allouche and Jeffrey O. Shallit. The ubiquitous prouhet-thue-morse sequence. In *Sequences and their Applications*.

[9] Jean-Paul Allouche and Jeffrey O. Shallit. *Automatic Sequences - Theory, Applications, Generalizations*. Cambridge University Press, 2003.

[10] Vikas G. Ashok and Ravi Mukkamala. A scalable and efficient privacy preserving global itemset support approximation using bloom filters. In *DBSec*, pages 382–389, 2014.

[11] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. Surroundsense: mobile phone localization via ambience fingerprinting. In *MOBICOM*, pages 261–272, 2009.

[12] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 136–145, 2001.

[15] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[16] T.-H. Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. Differentially private continual monitoring of heavy hitters from distributed streams. In *PETS*, pages 140–159, 2012.

[17] Graham Cormode. Data sketching. *Commun. ACM*, 60(9):48–55, August 2017.

[18] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[19] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS*, pages 218–231, 2012.

[20] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pages 285–304, 2006.

[21] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, pages 1–18, 2013.

[22] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[23] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *ACISP*, pages 261–278, 2017.

[24] Damien Desfontaines, Andreas Lochbihler, and David A. Basin. Cardinality estimators do not preserve privacy. In *PETS*.

[25] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security*, pages 303–320, 2004.

[26] Changyu Dong and Grigorios Loukides. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Trans. Information Forensics and Security*, 12(11):2792–2806, 2017.

[27] Cynthia Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.

[28] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, pages 486–503, 2006.

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

[30] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *ACISP*, pages 413–430, 2015.

[31] Tariq Elahi, George Danezis, and Ian Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In *ACM CCS*, pages 1068–1079, 2014.

[32] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In *ACM CCS*, pages 1054–1067, 2014.

[33] Ellis Fenske, Akshaya Mani, Aaron Johnson, and Micah Sherr. Distributed measurement with private set-union cardinality. In *ACM CCS*, pages 2295–2312, 2017.

[34] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[35] Oded Goldreich. *The Foundations of Cryptography*. Cambridge University Press, 2004.

[36] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford University Press, third edition edition, 2001.

[37] Hazar Harmouch and Felix Naumann. Cardinality estimation: An experimental survey. *PVLDB*, 11(4):499–512, 2017.

[38] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. Network-wide heavy hitter detection with commodity switches. In *SOSR*, 2018.

[39] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In *TCC*, pages 3–39, 2017.

[40] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *EDBT*, pages 683–692, 2013.

[41] Rob Jansen and Aaron Johnson. Safely measuring tor. In *ACM CCS*, pages 1553–1567, 2016.

[42] Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. In *CRYPTO*, pages 110–125, 2003.

[43] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS*, pages 830–842, 2016.

[44] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, pages 158–189, 2018.

[45] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography.*, pages 277–346. 2017.

[46] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear online complexity. In *ICALP*, pages 645–656, 2013.

[47] Akshaya Mani and Micah Sherr. Historϵ: Differentially private and robust statistics collection for tor. In *NDSS*, 2017.

[48] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. In *NDSS*, 2016.

[49] Darakhshan J. Mir, S. Muthukrishnan, Aleksandar Nikolov, and Rebecca N. Wright. Pan-private algorithms via statistics on sketches. In *PODS*, pages 37–48, 2011.

[50] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. In *USENIX Security*, 2001.

[51] A. B. M. Musa and Jakob Eriksson. Tracking unmodified smartphones using wi-fi monitors. In *SenSys*, pages 281–294, 2012.

[52] Nikos Ntarmos, Peter Triantafillou, and Gerhard Weikum. Counting at large: Efficient cardinality estimation in internet-scale data networks. In *ICDE*, 2006.

[53] Information Commisioner's Office. Wi-fi location analytics. 2016.

[54] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[55] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, pages 735–746, 2010.

[56] Nathaniel Schenker and Trivellore E. Raghunathan. Combining information from multiple surveys to enhance estimation of measures of health. *Statistics in medicine*, 26(8):1802–1811, 2007.

[57] Björn Scheuermann and Martin Mauve. Near-optimal compression of probabilistic counting sketches for networking applications. In *DIALM-POMC*, 2007.

[58] Berry Schoenmakers and Pim Tuyls. Efficient binary conversion for paillier encrypted values. In *EUROCRYPT*, pages 522–537, 2006.

[59] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.

[60] Hagen Sparka, Florian Tschorsch, and Björn Scheuermann. P2KMV: A privacy-preserving counting sketch for efficient and accurate set intersection cardinality estimations. *IACR Cryptology ePrint Archive*, 2018:234, 2018.

[61] Rade Stanojevic, Mohamed Nabeel, and Ting Yu. Distributed cardinality estimation of set operations with differential privacy. In *IEEE PAC*, pages 37–48, 2017.

[62] STASTICA. Marks & Spencer average weekly footfall in the United Kingdom (UK) 2009-2018. `https://www.statista.com/statistics/413515/marks-and-spencer-mands-average-weekly-footfall-united-kingdom-uk/`.

[63] Stephanie Clifford and Quentin Hardy. Attention, Shoppers: Store Is Tracking Your Cell. `https://www.nytimes.com/2013/07/15/business/attention-shopper-stores-are-tracking-your-cell.html`, 2013.

[64] The Guardian. Shops can track you via your smartphone, privacy watchdog warns. `https://www.theguardian.com/technology/2016/jan/21/shops-track-smartphone-uk-privacy-watchdog-warns`, 2016.

[65] Florian Tschorsch and Björn Scheuermann. An algorithm for privacy-preserving distributed user statistics. *Computer Networks*, 57(14):2775 – 2787, 2013.

[66] Wei Xi, Jizhong Zhao, Xiang-Yang Li, Kun Zhao, Shaojie Tang, Xue Liu, and Zhiping Jiang. Electronic frog eye: Counting crowd using wifi. In *INFOCOM*, pages 361–369, 2014.

[67] Qingjun Xiao, You Zhou, and Shigang Chen. Better with fewer bits: Improving the performance of cardinality estimation of large data streams. In *INFOCOM*, pages 1–9, 2017.

A List of Notations

- $\mathbb{F}_p$: the prime field.
- *DP*: the Data Parties.
- *CP*: the Computation Parties.
- n: the number of DPs.
- d: the number of CPs.
- N: the number of different elements in the set for the FM sketch.
- $\widetilde{N}$: the estimated value of N.
- m: the number of FM sketches we should use to improve the accuracy.
- w: the bit length of the FM-sketch
- $H : \{0,1\}^* \to \{0,1\}^{w-1}$: a hash function that maps an input uniformly to a $(w-1)$-bit string.
- $\rho : \{0,1\}^{w-1} \to [0, w-1]$: a function that takes a $(w-1)$-bit string as input and returns the number of trailing zeroes in it.
- ϕ: a correcting factor for FM sketches, which is 0.77351.
- κ: a correcting factor for FM sketches, which is 1.75.
- λ: a security parameter for statistical privacy, which can be set to be 40 at least.
- τ, the bit length of the plaintext domain.

B Protocols in Our Offline Phase

Here we provide the details of the offline protocols used by us. The protocols (10 – 12) were originally proposed in [20].

Protocol 10: *Rand*()

```
  Result: $[\![a]\!]$, where $a \in_R \mathbb{F}_p$
1 for $(i = 1; i \leq n; i++)$ do
3     Party $P_i$ samples $r_i \in_R \mathbb{F}_p$, and calls $\mathcal{F}_{[\![\cdot]\!]}$ with
        $(\mathsf{input}, r_i, P_i)$;
5     All parties obtains $[\![r_i]\!]$;
6 end
8 return $[\![a]\!] = \sum_{i=1}^{n} [\![r_i]\!]$
```

Protocol 11: *Rand*2()

```
   Result: $[\![b]\!]$ where $b \in_R \{0,1\}$
1  do
3      $[\![a]\!] \leftarrow Rand()$;
5      $[\![a^2]\!] \leftarrow [\![a]\!] \cdot [\![a]\!]$, reveal $a^2$;
6  while $a^2 = 0$;
8  $r = \sqrt{(a^2)}$;                          // $r \in_R \{a, -a\}$
10 $[\![c]\!] \leftarrow r^{-1} \cdot [\![a]\!]$;
12 return $[\![b]\!] \leftarrow 2^{-1} \cdot ([\![c]\!] + 1)$;
```

Protocol 12: *RandExp*(l)

```
   Input: $l$, a positive integer.
   Result: $([\![R^{-1}]\!], [\![R]\!], [\![R^2]\!], \cdots, [\![R^l]\!])$, where $R \in_R Z_p^*$
1  do
3      $[\![R]\!] \leftarrow Rand()$;
5      $[\![r]\!] \leftarrow Rand()$;
7      $[\![a]\!] \leftarrow [\![R]\!] \cdot [\![r]\!]$, reveal $a$;
8  while $a=0$;
10 $[\![R^{-1}]\!] \leftarrow a^{-1} \cdot [\![r]\!]$;
11 for $i = 2; i \leq l; i++$ do
13     $[\![R^i]\!] \leftarrow [\![R]\!] \cdot [\![R^{i-1}]\!]$;
14 end
16 return $([\![R^{-1}]\!], [\![R]\!], [\![R^2]\!], \cdots, [\![R^l]\!])$;
```

C SPDZ Ideal Functionality

We summarize the ideal functionality realized by SPDZ in Figure 7. it is taken from [44], and we will use it in our proof.

D Security Proof of the Offline Protocols

The offline protocols presented in Appendix B were developed before SPDZ. However, they can run on top of SPDZ and enjoy the strong security guarantees provided by SPDZ. Since we will use these protocols as sub-routines, here we provide the security proof of those protocols.

The ideal functionality of the offline protocols is shown in Figure 8. The functionality serves as a trusted blackbox generating authenticated random numbers. As a remark, since we are dealing with authenticated values, the ideal functionality has the same internal (i.e. a dictionary) and contains commands from the $\mathcal{F}_{\mathsf{SPDZ}}$ ideal functionality that are used to operate on authenticated values.

Functionality $\mathcal{F}_{\mathsf{SPDZ}}$

The functionality maintains a dictionary, Val, to keep track of the authenticated values. Entries of Val lie in the (fixed) finite field $\mathbb{F}_p$ and cannot be changed, for simplicity.

Commands for share authentication

Input: On receiving $(\mathsf{input}, id_1, \ldots, id_l, x_1, \ldots, x_l, P_j)$ where $x_i \in \mathbb{F}_p$ from party P_j and $(\mathsf{input}, id_1, \ldots, id_l, P_j)$ from all other parties, set $\mathsf{Val}[id_i] \leftarrow x_i$ for $1 \leq i \leq l$.
Linear comb.: On receiving $(\mathsf{LinComb}, \overline{id}, id_1, \ldots, id_t, c_1, \ldots, c_t, c)$, from all parties where $(id_1, \ldots, id_t) \in \mathsf{Val}.keys()$ and the combination co-coefficients $(c_1, \ldots, c_t, c) \in \mathbb{F}_p$, set $\mathsf{Val}[\overline{id}] \leftarrow \sum_{i=1}^{t} \mathsf{Val}[id_i] \cdot c_i + c$.
Reveal: On receiving (Reveal, id) from all parties, where $id \in \mathsf{Val}.keys()$, send $\mathsf{Val}[id]$, wait for x from the adversary, and output x to all parties.
Check: On receiving $(\mathsf{Check}, id_1, \ldots, id_t, x_1, \ldots, x_t)$ from every party P_i, wait for an input from the advery. If it inputs OK, and $\mathsf{Val}[id_j] = x_j$ for all j, return OK to all parties, otherwise return $\perp$ and terminate.
Abort: On receiving **Abort** from the adversary, send $\perp$ to all parties and terminate.

Commands for pre-processing

Input Tuple: On receiving $(\mathsf{InputTuple}, P_j, id)$ from all parties, sample $r \in_R \mathbb{F}_p$, set $\mathsf{Val}[id_j] \leftarrow r$ and output it to P_j.
Triple: On receiving $(\mathsf{Triple}, id_a, id_b, id_c)$ from all parties, sample $a, b \in_R \mathbb{F}_p$ and set $(\mathsf{Val}[id_a], \mathsf{Val}[id_b], \mathsf{Val}[id_c]) \leftarrow (a, b, a \cdot b)$

Commands for online computation

Initialize: On input $(\mathsf{Init}, \mathbb{F}_p)$ from all parties, store $\mathbb{F}_p$.
Input: On receiving $(\mathsf{Input}, P_i, id, x)$ from p_i and $(\mathsf{Input}, P_i, id)$ from all other parties, with id a fresh identifier and $x \in \mathbb{F}_p$, store $\mathsf{Val}[id] \leftarrow x$.
Add: On receiving $(\mathsf{Add}, id_1, id_2, id_3)$ from all parties where $(id_1, id_2 \in \mathsf{Val}.keys())$ and id_3 is a fresh identifier, set $\mathsf{Val}[id_3] \leftarrow \mathsf{Val}[id_1] + \mathsf{Val}[id_2]$.
Multiply: On receiving $(\mathsf{Mult}, id_1, id_2, id_3)$ from all parties where $(id_1, id_2 \in \mathsf{Val}.keys())$ and id_3 is a fresh identifier, set $\mathsf{Val}[id_3] \leftarrow \mathsf{Val}[id_1] \cdot \mathsf{Val}[id_2]$.
Output: On receiving (Output, id) from all honest parties (where $id \in \mathsf{Val}.keys()$), retrieve $y \leftarrow \mathsf{Val}[id]$ and output it to the adversary. Wait for an input from the adversary. If this is Deliver then output y to all parties, otherwise abort.

Figure 7: Ideal functionality realized by SPDZ

The theorem is stated below in Theorem 6. The proof follows the theorem. Because we use the UC model, we can take the advantage of the universal composability and prove security in $\mathcal{F}$-hybrid model [45]. That is, instead of proving the indistinguishability between a real world execution and ideal world execution, we construct a

Functionality $\mathcal{F}_{\mathsf{offline}}$

The functionality maintains a dictionary, Val, to keep track of the authenticated values. Entries of Val lie in the (fixed) finite field $\mathbb{F}_p$ and cannot be changed, for simplicity. The functionality includes all commands in $\mathcal{F}_{\mathsf{SPDZ}}$
Rand: On receiving (rand, id) from all parties, sample a uniform $r \leftarrow F_p$ and set $\mathsf{Val}[id] \leftarrow r$.
Rand2: On receiving $(\mathsf{rand2}, id)$ from all parties, sample a uniform $r \leftarrow \{0,1\}$ and set $\mathsf{Val}[id] \leftarrow r$.
RandExp: On receiving $(\mathsf{randexp}, l, id_0, id_1, \ldots, id_l)$ from all parties, sample a uniform $r \leftarrow F_p$ and store $r^{-1}, r, r^2, \cdots, r^l$ in $\mathsf{Val}[id_0], \mathsf{Val}[id_1], \cdots, \mathsf{Val}[id_l]$ respectively.

Figure 8: offline Ideal functionality

hybrid protocol in which all invocations to the SPDZ sub-routines are replaced by calls to the SPDZ ideal functionality, then prove the execution of this hybrid protocol and the ideal world execution are indistinguishable. Since the security of SPDZ has already been proved, the indistinguishability between the real and ideal executions is entailed by the UC theorem.

Theorem 6. *In the $\mathcal{F}_{SPDZ}$-hybrid model, the protocol $\Pi_{offline}$ implements $\mathcal{F}_{offline}$ against any static active adversary corrupting up to $n-1$ (computation) parties.*

Proof. We show how to construct an ideal world simulator $\mathcal{S}_{\mathsf{offline}}$ so that a PPT adversary (or environment) $\mathcal{Z}$ cannot distinguish between whether it is executing the hybrid version of Π_{offline} or is interacting with the simulator who has access to the ideal functionality $\mathcal{F}_{\mathsf{offline}}$. The adversary is static, which means it chooses the parties to corrupt at the very beginning, and is active, which means it can deviate arbitrarily from the protocols. The simulator runs Π_{offline} with the adversary and simulates all SPDZ ideal functionality internally. It simulates the same interface that $\mathcal{Z}$ will see when interacting with a real protocol. The specification of the simulator $\mathcal{S}_{\mathsf{offline}}$ is as the following:

Simulator $\mathcal{S}_{\mathsf{offline}}$

Let H denote the set of honest parties and A the complement thereof.
Rand:

1. On receiving $(\mathsf{input}, id_i, x_i, P_i)$ from the adversary for all $P_i \in A$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**input**.
2. On receiving $(\mathsf{Add}, id'_1, \cdots, id'_n, id)$ from the adversary for $P_i \in A$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Add**.
3. Call $\mathcal{F}_{\mathsf{offline}}$.**Rand** with input $(Rand, id)$.

Rand2:

1. Run simulation for **Rand** as stated above, at the end, call $\mathcal{F}_{\mathsf{offline}}$.**Rand** with input (Rand, id_a).
2. Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Multiply** on receiving $(\mathsf{Mult}, id_a, id_a, id_{a^2})$ from the Adversary for all $P_i \in A$. At this point, the simulator does not have the value for *id* in its emulated SPDZ ideal functionality, so it cannot compute the value for id_{a^2}, i.e. a^2. It notes this down and puts a dummy value there for id_{a^2}, then calls $\mathcal{F}_{\mathsf{offline}}$.**Multiply** with input $(\mathsf{Mult}, id_a, id_a, id_{a^2})$.
3. One Receiving $(\mathsf{Reveal}, id_{a^2})$ from all Adversary $P_i \in A$, call $\mathcal{F}_{\mathsf{offline}}$.**Reveal** with input $(\mathsf{Reveal}, id_{a^2})$ and obtains the value of a^2, then replace the dummy value in the SPDZ ideal functionality for id_{a^2} with a^2, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Reveal** with the value.
4. If $a^2 = 0$, repeat the above steps, otherwise compute locally $r = \sqrt{a^2}$ and r^{-1} and continue.
5. On receiving $(\mathsf{LinComb}, id_c, id_a, r^{-1})$ from the adversary for all $P_i \in A$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb**, and call $\mathcal{F}_{\mathsf{offline}}$.**Linearcomb** with $(\mathsf{LinComb}, id_c, id_a, r^{-1})$, (i.e. compute $c = r^{-1}a$ and stores the result in id_c – a dummy value for now in the emulated SPDZ ideal functionality).
6. On receiving $(\mathsf{LinComb}, id_b, id_c, 2^{-1}, 2^{-1})$ from the adversary for all $P_i \in A$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb**, and call $\mathcal{F}_{\mathsf{offline}}$.**Linearcomb** with $(\mathsf{LinComb}, id_b, id_c, 2^{-1}, 2^{-1})$, (i.e. compute $b = 2^{-1}(c+1)$) and store the result in id_b – a dummy value for now in the emulated SPDZ ideal functionality).

RandExp:

1. Run simulation for **Rand** as stated above, at the end of each run, call $\mathcal{F}_{\mathsf{offline}}$.**Rand** with input (Rand, id_R) and (Rand, id_r).
2. Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Multiply** on receiving $(\mathsf{Mult}, id_R, id_r, id_a)$ with a dummy value there for id_a, then calls $\mathcal{F}_{\mathsf{offline}}$.**Multiply** with input $(\mathsf{Mult}, id_R, id_r, id_a)$.
3. One Receiving (Reveal, id_a) from all Adversary $P_i \in A$, call $\mathcal{F}_{\mathsf{offline}}$.**Reveal** with input (Reveal, id_a) and obtains the value of a, then replace the dummy value in the SPDZ ideal functionality for id_a with a, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Reveal** with the value.
4. If $a = 0$ repeat the above steps. Otherwise compute a^{-1} and continue.
5. On receiving $(\mathsf{LinComb}, id_{R^{-1}}, id_r, a^{-1})$ from the adversary for all $P_i \in A$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb**, and call $\mathcal{F}_{\mathsf{offline}}$.**Linearcomb** with $(\mathsf{LinComb}, id_{R^{-1}}, id_r, a^{-1})$, (i.e. compute $R^{-1} = a^{-1}r$ and stores the result in $id_{R^{-1}}$ – a dummy value for now in the emulated SPDZ ideal functionality).
6. For $2 \le i \le l$, on receiving $(\mathsf{Mult}, id_R, id_{R^{i-1}}, id_{R^i})$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Multiply** with a dummy value there for id_{R^i}, then calls $\mathcal{F}_{\mathsf{offline}}$.**Multiply** with input $(\mathsf{Mult}, id_R, id_{R^{i-1}}, id_{R^i})$.

As we can see, the simulator can perfectly simulate the hybrid execution, in which the adversary's view consists of the SPDZ ideal functionality calls and output if any from the ideal functionality (e.g. **Reveal**). The view of the adversary in the simulated execution and the hybrid execution are identically distributed. In various places in the simulation, the simulator puts a dummy value as a place holder in it emulated $\mathcal{F}_{\mathsf{SPDZ}}$ functionality. If the value is never revealed, then this is fine. But if the value needs to be revealed, the simulator calls the **Reveal** command of the $\mathcal{F}_{\mathsf{offline}}$ functionality to get the value, and replace the dummy value with the right one, so that in the joint outputs will be consistent. When executing the protocol, the adversary can invoke **Check** or **Abort**, the simulator calls the same command in $\mathcal{F}_{\mathsf{offline}}$ to handle it. □

E Alternative Protocol for Extracting Estimator

Recall that z_N is the index of the first 0 bit in a sketch, thus extracting z_N can be converted to a search problem. Protocol 13 performs essentially a binary search. In Protocol 13, the bits in the sketch are first negated (lines 3 - 5). Then the sketch is divided into two halves. If all bits now in the first half are 0, then before negation, all of them were 1, which means the first 0 we are looking for is in the second half. Then we know z_N must be the size of the first half plus some offset into the second half, and we can throw away the first half and do a binary search on the second half to find the offset. If not all bits in the first half are 0, then the first 0 we are looking for is in the first half. Then we can throw away the second half and do another binary search on the first half. Obviously, we cannot reveal whether the first half is all 0 in the protocol, as this leaks information. So what we do is to sum all bits in the first half into x, then interpolate a lookup polynomial f such that $B_0 = f(x+1) = 1$ if $x = 0$ and 0 otherwise (lines 8 – 10). Then we obliviously combine the first half and the second half, by multiplying every bit in the second half with B_0 and add the result to the first half (lines 12 – 17). Since the multiplications are independent, they can be batched together. Note also that an extra addition is needed if the two halves are not of the same size. If the first half is all 0, then we need to continue searching the second half. In this case, B_0 is 1 and what we get after the addition is the second half. If the first half is not all 0, then we do not have to search the second half at all. In this case B_0 is 0 and we get the first half after the addition. Then we start the while loop again until there are only few bits left to search. In this case, we take the bits left and do a lookup to finish the search (lines 20 – 22). There are $\log(w)$ iterations in the while loop, and in each iteration, we need two rounds: one round for line 10 (because of the multiplication in the *Lookup* protocol) and one round for the multiplications in the for loop staring at line 12.

F Proof of Theorem 1

Note in the theorem, we use the notion of statistical security. This is because of the ZeroTest protocol (Protocol 5). More specifically, in line 1 – 2, m is computed from $[\![r]\!]$ and $[\![a]\!]$, and the value of m is revealed. The distribution of m and that of r are not identical, therefore we need to ensure the difference is negligible in a security parameter. The statistical security is ensured by the following Lemma [58]:

Lemma 8. *Let M and K be positive integers with $M \leq K$. Let X, U be random variables in $[0,\cdots,M-1]$, $[0,\cdots,K-1]$ respectively such that U is uniform. Then $\Delta(U;X+U) \leq (M-1)/K$ and this bound is tight.*

Essentially we achieve this by requiring r to be much larger than a. Recall that in the protocol a is an integer from $[0, 2^{\tau}-1]$ and r is and integer from $[0, 2^{\tau+\kappa}-1]$, therefore by the above lemma the statistical distance between the two distribution is at most $2^{-\kappa}$. This guarantees the simulated execution and the real execution is statistically indistinguishable up to the security parameter κ.

Now we are ready to prove Theorem 1.

Proof. We construct a simulator $\mathcal{S}_{\mathsf{PDCE}}$ such that a poly-time environment $\mathcal{Z}$ cannot distinguish between the execution of the hybrid protocol and the execution in the ideal world. The specification of the simulator $\mathcal{S}_{\mathsf{PDCE}}$ is as the following:

Protocol 13: *ExtractZBS*$([\![\mathsf{BFS}^1_{\cup}[0]]\!],\cdots,[\![\mathsf{BFS}^1_{\cup}[w-1]]\!],\cdots,[\![\mathsf{BFS}^m_{\cup}[0]]\!],\cdots,[\![\mathsf{BFS}^m_{\cup}[w-1]]\!])$

```
Input: $[\![\mathsf{BFS}^1_{\cup}[0]]\!],\cdots,[\![\mathsf{BFS}^1_{\cup}[w-1]]\!],\cdots,[\![\mathsf{BFS}^m_{\cup}[0]]\!],\cdots,$
       $[\![\mathsf{BFS}^m_{\cup}[w-1]]\!]$, the shares of the $m$ binary FM sketches
Result: $Z_N$, the estimator extracted from the sketches
1  $[\![Z_N]\!] = 0$;
2  for $i = 1; i \leq m; i++$ do
3      for $j = 0;\ j \leq w-1;\ j{+}{+}$ do
4          $[\![\mathsf{BFS}^i_{\cup}[j]]\!] = 1 - [\![\mathsf{BFS}^i_{\cup}[j]]\!]$ ;      // negate the bit
5      end
6      $size = w, t = \lceil \frac{size}{2} \rceil, [\![z_{N,i}]\!] = 0$;
       // binary search until not worth it
7      while $size > 3$ do
8          $[\![x+1]\!] = 1 + \sum_{l=0}^{t-1} [\![\mathsf{BFS}^i_{\cup}[l]]\!]$ ;
           // interpolate the lookup polynomial
9          $(t, \beta_0, \cdots, \beta_t) \leftarrow interpolate()$;
           // $B_0 = 1$ if $x = 0$, $B_0 = 0$ otherwise
10         $[\![B_0]\!] = Lookup([\![x+1]\!], t, \beta_0, \cdots, \beta_t)$;
11         $[\![z_{N,i}]\!] = [\![z_{N,i}]\!] + t \cdot [\![B_0]\!]$;
12         for $j = 0;\ j < size - t;\ j{+}{+}$ do
13             $[\![\mathsf{BFS}^i_{\cup}[j]]\!] = [\![\mathsf{BFS}^i_{\cup}[j]]\!] + [\![B_0]\!] \cdot [\![\mathsf{BFS}^i_{\cup}[j+t]]\!]$ ;
14         end
15         if $size$ is odd then
16             $[\![\mathsf{BFS}^i_{\cup}[size-t]]\!] = [\![\mathsf{BFS}^i_{\cup}[size-t]]\!] + [\![B_0]\!]$;
17         end
18         $size = t, t = \lceil \frac{size}{2} \rceil$;
19     end
20     $[\![x]\!] = 1 + \sum_{i=0}^{size-1} 2^i \cdot [\![\mathsf{BFS}^i_{\cup}[size-1]]\!]$;
       // interpolate the lookup polynomial
21     $(2^{size}, \beta_0, \cdots, \beta_{2^{size}}) \leftarrow interpolate()$;
       // final lookup for the rest of the bits
22     $[\![z_{N,i}]\!] = [\![z_{N,i}]\!] + Lookup([\![x]\!], 2^{size}, \beta_0, \cdots, \beta_{2^{size}})$;
23     $[\![Z_N]\!] = [\![Z_N]\!] + [\![z_{N,i}]\!]$;
24 end
25 return $Z_N \leftarrow \mathsf{Output}([\![Z_N]\!])$;
```

Simulator $\mathcal{S}_{\mathsf{PDCE}}$

Let H denote the set of honest parties and A the complement thereof. When executing the online protocol, the offline part has already been done and necessary values have been generated prior to the start, but to make it clear we show which offline protocol is used and which values are generated in the simulation.

Share:

1. On receiving (rand, id_a) from all CPs in A, emulate $\mathcal{F}_{\mathsf{offline}}$.**Rand** by sampling a random $a \leftarrow F_p$ and setting $\mathsf{Val}[id_a] \leftarrow a$.
2. On receiving (Reveal, id) from all CPs in A, take the value $\mathsf{Val}[id] = a$ and creates shares of $[\![a]\!]$, and send them to the DP only.
3. If the DP is in A, wait for it to broadcast $x' - a$, then compute $x' = x' - a + a$. If the DP is in H, then x is its input, which is

not known by the simulator. However the simulator can simulate it by choosing a random $x' - a$ and broadcast it.

4. On receiving $(\mathsf{Linearcomb}, id_x, (x'-a))$ from all CPs in A, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb** to compute $a + x' - a$.

5. Call $\mathcal{F}_{\mathsf{PDCE}}$.**Share** with (share, id_x) for all CPs in A. If the DP is in A, also call $\mathcal{F}_{\mathsf{PDCE}}$.**Share** with $(\mathsf{share}, id_x, x')$. It should be clear that if the DP is honest, $\mathcal{F}_{\mathsf{PDCE}}$ will set $\mathsf{Val}[id_x] \leftarrow x$ and if the DP is corrupted, $\mathcal{F}_{\mathsf{PDCE}}$ will set $\mathsf{Val}[id_x] \leftarrow x'$ with the value chosen by the adversary.

Data Collection:

1. For each DP_i, run simulation for **Share** $m \cdot w$ times to share its $\overline{\mathsf{OFS}}$ with CPs.
2. For each DP_i, run simulation for **Share** $m \cdot w$ times to share its OFS with CPs.
3. Then send (go) to $\mathcal{F}_{\mathsf{SPDZ}}$ on behalf of each honest party.

MergeShare:

1. Emulating $\mathcal{F}_{\mathsf{SPDZ}}$.**Add** for all additions needed in the MergeShare protocol. Make sure the id's of the addends are valid in the process and record the id's of the sums and the addends. After all finished, put the id's in the correct positions of $id^{\overline{\mathsf{OFS}}}, id^{\mathsf{OFS}}, id^{\mathsf{FS}_{\cup}}$
2. Call $\mathcal{F}_{\mathsf{PDCE}}$.**MergeShare** with $(\mathsf{mergeshare}, id^{\overline{\mathsf{OFS}}}, id^{\mathsf{OFS}}, id^{\mathsf{FS}_{\cup}})$ for all CPs in A.

Lookup:

1. Emulate $\mathcal{F}_{\mathsf{offline}}$.**RandExp** on receiving $(\mathsf{randexp}, l, id_{r^{-1}}, id_r, \ldots, id_{r^l})$ from all CPs in A.
2. Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Multiply** on receiving $(\mathsf{Mult}, id_x, id_{r^{-1}}, id_a)$ from all CPs in A.
3. On receiving (Reveal, id_a) from all CPs in A, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Reveal** with a random a.
4. For $2 \le i \le l$, compute a^i, on receiving $(\mathsf{linearcomb}, id_{x^i}, id_{r^i}, a^i)$ from all CPs in A, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb**.
5. On receiving $(\mathsf{linearcomb}, id_y, id_{x^0}, \ldots id_{x^l}, \beta_0, \beta_l)$, from all CPs in A, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb**.
6. Call $\mathcal{F}_{\mathsf{PDCE}}$.**lookup** with $(\mathsf{lookup}, id_x, id_y, \beta_0, \beta_l)$ for all CPs in A.

ZeroTest:

1. For $0 \le i \le l-2$, on receiving $(\mathsf{rand2}, id_{r_i})$ from all CPs in A, emulate $\mathcal{F}_{\mathsf{offline}}$.**Rand2**.
2. On receiving $(\mathsf{linearcomb}, id_r, id_{r_0}, \cdots, id_{r^{l-2}}, 2^{l-2}, \cdots, 2^0)$ from all CPs in A, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb**.
3. Interpolate the lookup polynomial and obtains $(\tau, \beta_0, \cdots, \beta_\tau)$.
4. On receiving $(\mathsf{Add}, id_r, id_a, id_m)$, emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Add**.
5. On receiving (Reveal, id_m), emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Reveal** with a random m.
6. Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb** for the Hamming distance computation, with result in id_{1+h}.
7. Run simulation for *Lookup* with id_{1+h} and $\beta_0, \cdots, \beta_\tau$ with the result in id_b
8. Call $\mathcal{F}_{\mathsf{PDCE}}$.**ZeroTest** with $(\mathsf{zerotest}, id_a, id_b)$ for all CPs in A.

ToBinary:

1. For $1 \le j \le m$ and $0 \le l \le w-1$, run the simulation for ZeroTest, each on $id_{\mathsf{FS}^j_{\cup}[l]}$ (as id_a) and store the result in $id_{\mathsf{BFS}^j_{\cup}[l]}$ (as id_b).

ExtractZ:

1. Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Add** for $Z_N = \sum_{i=1}^m \mathsf{BFS}^i_{\cup}[0]$
2. Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Add** and $\mathcal{F}_{\mathsf{SPDZ}}$.**multiply** for the iterations, in which $\mathsf{BFS}^i_{\cup}[l] = \mathsf{BFS}^i_{\cup}[l-1] \cdot \mathsf{BFS}^i_{\cup}[l]$ then $Z_N = z_{N,i} + \mathsf{BFS}^i_{\cup}[l]$, where $1 \le l \le w-1$ and $1 \le i \le w-1$. The final result is in id_{Z_N}.
3. Call $\mathcal{F}_{\mathsf{PDCE}}$.**ExtractZ** with $(\mathsf{extractz}, id_{\mathsf{BFS}^1_{\cup}[0]}, \cdots, id_{\mathsf{BFS}^1_{\cup}[w-1]}, \cdots, id_{\mathsf{BFS}^m_{\cup}[0]}, \cdots, id_{\mathsf{BFS}^m_{\cup}[w-1]}, id_{Z_N})$. Then emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Output** with id_{Z_N}.

ExtractZBS:

1. Do the following m times, in the ith iteration:
   (a) Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb** for negating the bits, the results are in $id_{\mathsf{BFS}^i_{\cup}[j]}$ for $0 \le j \le w-1$.
   (b) Set $size = w, t = \lceil \frac{size}{2} \rceil$ then while $size > 3$, do the following:
   Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb** and $\mathcal{F}_{\mathsf{SPDZ}}$.**Add** for computing $x + 1 = 1 + \sum_{i=0}^{w-1} \mathsf{BFS}^i_{\cup}[j]$, the result is in id_{x+1}. Then interpolate the lookup polynomial to get $\beta_0, \cdots, \beta_t$, and run the simulation for Lookup with id_{x+1} and $\beta_0, \cdots, \beta_t$, the result is stored in id_{B_0}.
   Then emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb** to compute $z_{N,i} = z_{N,i} + t \cdot B_0$ with result stored in id_z. Then emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Add** and $\mathcal{F}_{\mathsf{SPDZ}}$.**Multiply** for computing $\mathsf{BFS}^i_{\cup}[j] = \mathsf{BFS}^i_{\cup}[j] + B_0 \cdot \mathsf{BFS}^i_{\cup}[j+t]$ for $0 \le j < size - t$, and if $size$ is odd, compute also $\mathsf{BFS}^i_{\cup}[size-t] = \mathsf{BFS}^i_{\cup}[size-t] + B_0$. Then set $size = t$ and $t = \lceil \frac{size}{2} \rceil$.
   (c) Emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Linearcomb** for computing $x = 1 + \sum_{i=0}^{size-1} 2^i \mathsf{BFS}^i_{\cup}[size-1]$ with the result in id_x. Then interpolate the lookup polynomial to get $\beta_0, \cdots, \beta_{2^{size}}$.
   (d) Simulate Lookup with $id_x, 2^{size}, \beta_0, \cdots, \beta_{2^{size}}$ with the result in id_b, then emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Add** for computing $z_{N,i} = z_{N,i} + b$, then $Z_N = Z_N + z_{N,i}$, store the result in id_{Z_N}.
2. After all finished, take the last id_{Z_N} then call $\mathcal{F}_{\mathsf{PDCE}}$.**ExtractZ**, with $(\mathsf{extractz}, id_{\mathsf{BFS}^1_{\cup}[0]}, \cdots, id_{\mathsf{BFS}^1_{\cup}[w-1]}, \cdots, id_{\mathsf{BFS}^m_{\cup}[0]}, \cdots, id_{\mathsf{BFS}^m_{\cup}[w-1]}, id_{Z_N})$. Then emulate $\mathcal{F}_{\mathsf{SPDZ}}$.**Output** with id_{Z_N}.

Corrupt:

1. On receiving $(\mathsf{Corrupt}, DP_i)$ from the adversary, move DP_i from H to A. Then call $\mathcal{F}_{\mathsf{PDCE}}$.**Corrupt** with $(\mathsf{Corrupt}, DP_i)$.

Abort:

1. On receiving (Abort) from the adversary, call $\mathcal{F}_{\mathsf{PDCE}}$.**Abort** with (Abort) and terminate. If an honest party calls $\mathcal{F}_{\mathsf{PDCE}}$.**Abort** with (Abort), also send (Abort) to the adversary in the name of that honest party then terminate.

□

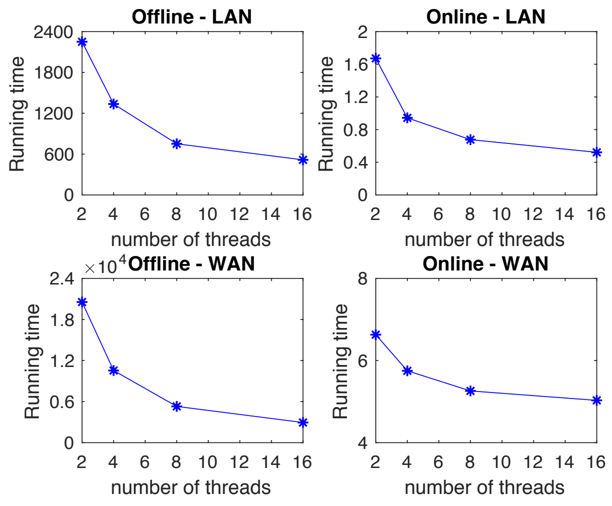


Figure 9: Running time with different numbers of threads: 5 CPs (16 threads at most), $n = 10^9, m = 4000$, 20 DPs

G Additional Experimental Results

In Table 5, we show the performance of a DP updating its sketches. In the experiment, we used different number of sketches, from 1000 to 4000. We use $w = 24$ for all sketches. We measure the total time for adding 10^6 elements into the m sketches. The experiment ran with 4 threads on a quadcore desktop (i7-6700k CPU and 16 GB RAM). As we can see, the time for building is reasonable. Even with $m = 4000$, the total time is just about 30 seconds. This means a DP with such a hardware configuration can handle 33,244 updates per second, which should be high enough for most applications.

m	1000	2000	4000
Time (s)	7.64	15.25	30.08
Throughput (updates/s)	130,890	65,573	33,244

Table 5: Performance: adding 10^6 observations into sketches

Next we show the CPs' running time with different numbers of threads. Each CP is a R5.4xLarge instance in Amazon AWS cloud, with 16 vCPUs (8 physical cores). In the experiment, we ran the CPs with 2, 4, 8, 16 threads. As we can see in Fig. 9, multi-threading is effective in reducing the total running time. However, generally the benefit of using more threads diminishes gradually when approaching the physical limit.

In Table 6, we show the performance of our protocol with p of different sizes. The prime number p decides the underlying finite field we use in the secure computation. Its size is dependent on the size of the plaintext domain τ and the statistical security parameter λ. For example, if we want to increase the statistical security parameter, then we need to use a larger p. As we can see, the size of p does have a significant impact on the performance of the offline computation. This is because in the SPDZ pre-processing protocol, somewhat homormophic encryption is used, whose performance is sensible to the size of p. Previously, the largest size of p we used was 70-bit. When the size of p increases to 160-bit, the running time of the offline phase increases to almost 4 times in LAN, and almost 2 times in WAN. The online protocol however is less affected. The communication cost increases as well when the size of p increases.

			$l = 70$	$l = 110$	$l = 160$
Running Time (s)	LAN	Offline	515.8	1167.7	2088.0
		Online	0.522	0.598	0.733
	WAN	Offline	2944.2	4162	5672.2
		Online	5.03	5.25	5.52
Communication (GB)	Offline		73.7	149.6	226.6
	Online		0.121	0.125	0.168

Table 6: Performance with different p:5 CPs (16 threads), $N = 10^9, m = 4000$, 20 DPs

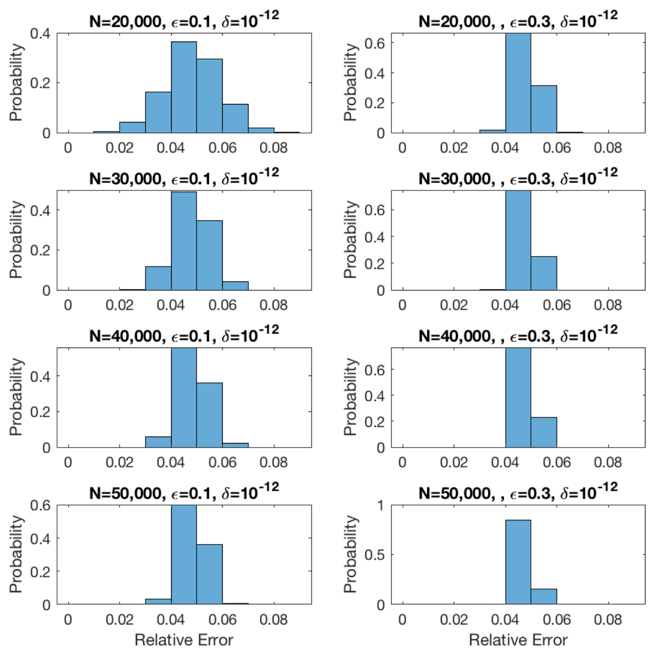


Figure 10: Distribution of relative errors of the protocol in [33]

In Figure 10, we show the distribution of relative errors of the estimations made using the protocol in [33]. The errors come from two sources: (1) Hash collision. The protocol in [33] hashes element into a hashtable and there is a probability of collision. The authors use a hashtable of size $1/f$ times the maximum size of the set to be measured, where f is a threshold collision probability. The default $f = 0.1$ means the collision probability will be less than 0.1, and accordingly the hashtable size is set to 10 times of the input size. (2) (ε, δ)-differential privacy. The protocol adds binomial noise to the data. The number of noise bits to be added is $\left\lceil \frac{64 \ln(2/\delta)}{\varepsilon^2} \right\rceil$, which is 20,142 when $\varepsilon = 0.3, \delta = 10^{-12}$ and 181,275 when $\varepsilon = 0.1, \delta = 10^{-12}$. We can see that the relative errors are distributed around 4%. The errors become more concentrated, when the input size increases. This could be because the amount of noise added for differential privacy is constant, therefore the errors become relatively smaller when the cardinality becomes smaller. The maximum relative error is larger than that of our protocol when $m = 4000$.

H Analysis of $f(y)$

$$\begin{aligned}f(y) =&f_1(y)-f_2(y)\\ =&b^4-by+(b-b^5-b^2)y^2+b^3y^3+(b^2-b^6-b^4)y^4+b^5y^5\\ &+(b^6-b^3+b^7)y^6-b^7y^7\\ =&-b(y-1)\Big(b^6y^6+(b^2-b^5)y^5+(b^2-b^5-b^4)y^4\\ &+(b^2+b^3-b^4-b)y^3+(b^3-b^4-b)y^2+(b^3-1)y+b^3\Big)\end{aligned}$$

In the above, $b=e^{-2^{-c}}$ and $y=e^{-2^{-t-c}}$, where c and t both are integers and $c\in(0,+\infty)$ and $t\in(0,+\infty)$. Therefore $b\in(e^{-1},1)$ and $y\in(b,1)$.

Theorem 7. *When $b\in(e^{-1},1)$, $f(y)<0$ for all $y\in(b,1)$*

Proof. In $f(y)$, $b>0$ and $y-1<0$, therefore the term $-b(y-1)>0$. To prove the theorem it is sufficient to show

$$\begin{aligned}g(y) =&b^6y^6+(b^2-b^5)y^5+(b^2-b^5-b^4)y^4+(b^2+b^3-b^4-b)y^3\\ &+(b^3-b^4-b)y^2+(b^3-1)y+b^3\\ <&0\end{aligned}$$

in the given range. To do so, we analyse its derivative, from the 6-th derivative backwards. The derivatives are:

$$\begin{aligned}g'(y) =&6b^6y^5+5(b^2-b^5)y^4+4(b^2-b^5-b^4)y^3+3(b^2+b^3-b^4-b)y^2\\ &+2(b^3-b^4-b)y+b^3-1\\ g''(y) =&30b^6y^4+20(b^2-b^5)y^3+12(b^2-b^5-b^4)y^2+6(b^2+b^3-b^4-b)y\\ &+2(b^3-b^4-b)\\ g^{(3)}(y) =&120b^6y^3+60(b^2-b^5)y^2+24(b^2-b^5-b^4)y+6(b^2+b^3-b^4-b)\\ g^{(4)}(y) =&360b^6y^2+120(b^2-b^5)y+24(b^2-b^5-b^4)\\ g^{(5)}(y) =&720b^6y+120(b^2-b^5)\\ g^{(6)}(y) =&720b^6\end{aligned}$$

1. It easy to see that $g^{(6)}(y)>0$ for all $b\in(e^{-1},1)$ $y\in(b,1)$. Therefore $g^{(5)}(y)$ increases monotonically in the same range.
2. Since for all $b\in(e^{-1},1)$ $g^{(5)}(y)$ increases monotonically when $y\in(b,1)$, let us first check $g^{(5)}(b)$:

   $$g^{(5)}(b)=720b^7-120b^6+120b^3=120b^3(6b^4-b^3+1)$$

   We can see that when $b\in(e^{-1},1)$, both $120b^3>0$ and $(6b^4-b^3+1)>0$, then $g^{(5)}(b)>0$. Since $g^{(5)}(b)>0$, for all $b<y<1$, $g^{(5)}(y)>0$. Therefore $g^{(4)}(y)$ increases monotonically in the same range.
3. By Lemma 9, $g^{(4)}(b)>0$ for all $b\in(e^{-1},1)$, then $g^{(3)}(y)$ increases monotonically in $y\in(b,1)$
4. By Lemma 10, $g^{(3)}(b)>0$ for all $b\in(e^{-1},1)$, then $g''(y)$ increases monotonically in $y\in(b,1)$
5. By Lemma 11, $g''(b)>0$ for all $b\in(0.552466,1)$, and $g''(b)<0$ for all $b\in(e^{-1},0.552466)$, $g''(1)>0$ for all $b\in(e^{-1},1)$.
   (a) when $b\in(0.552466,1)$, $g''(b)>0$, $g''(1)>0$, as $g''(y)$ increases monotonically in $y\in(b,1)$, then $g''(y)>0$ for all $y\in(b,1)$, then $g'(y)$ increases monotonically in $y\in(b,1)$. By Lemma 12, $g'(b)<0$ for all $b\in(e^{-1},1)$, $g'(1)<0$ for all $b\in(e^{-1},0.594173)$ and $g'(1)>0$ for all $b\in(0.594173,1)$.
      i. when $b\in(0.552466,0.594173)$, we know $g'(b)<0$ and $g'(1)<0$, as $g'(y)$ increases monotonically in $y\in(b,1)$, then $g'(y)<0$ for all $y\in(b,1)$. Then $g(y)$ increases monotonically in $y\in(b,1)$. By Lemma 13, $g(b)<0$ for all $b\in(e^{-1},1)$, and $g(1)<0$ for all $b\in(e^{-1},1)$, then $g(y)<0$ for all $y\in(b,1)$, which completes the proof when $b\in(0.552466,0.594173)$.
      ii. when $b\in(0.594173,1)$, we know $g'(b)<0$ and $g'(1)>0$, as $g'(y)$ increases monotonically in $y\in(b,1)$, then there exists a $y_0\in(b,1)$, which makes $g'(y_0)=0$, and $g'(y)<0$ for $y\in(b,y_0)$ and $g'(y)>0$ for $y\in(y_0,1)$. Then $g(y)$ decreases monotonically in $y\in(b,y_0)$ and increases monotonically in $y\in(y_0,1)$. By Lemma 13, $g(b)<0$ for all $b\in(e^{-1},1)$, and $g(1)<0$ for all $b\in(e^{-1},1)$, then $g(y)<0$ for all $y\in(b,1)$, which completes the proof when $b\in(0.552466,0.594173)$.
   (b) when $b\in(e^{-1},0.552466)$, $g''(b)<0$, $g''(1)>0$, as $g''(y)$ increases monotonically in $y\in(b,1)$, then there exists a $y_1\in(b,1)$, which makes $g''(y_1)=0$, and $g''(y)<0$ for $y\in(b,y_1)$ and $g''(y)>0$ for $y\in(y_1,1)$. Then $g'(y)$ decreases monotonically in $y\in(b,y_1)$ and increases monotonically in $y\in(y_1,1)$. By Lemma 12, $g'(b)<0$ for all $b\in(e^{-1},1)$, and $g'(1)<0$, then $g'(y)<0$ for all $y\in(b,1)$. Then $g(y)$ decrease monotonically in $y\in(b,1)$. By Lemma 13, $g(b)<0$ for all $b\in(e^{-1},1)$, and $g(1)<0$ for all $b\in(e^{-1},1)$, then $g(y)<0$ for all $y\in(b,1)$, which completes the proof for $b\in(e^{-1},0.552466)$.

□

Lemma 9. $g^{(4)}(b)=24b^2(15b^6-5b^4-b^3-b^2+5b+1)>0$ *for all $b\in(e^{-1},1)$*

Proof. Let $h_4(b)=15b^6-5b^4-b^3-b^2+5b+1$, then $h_4'(b)=90b^5-20b^3-3b^2-2b+5$, $h_4''(b)=450b^4-60b^2-6b-2$, and $h_4^{(3)}(b)=1800b^3-120b-6$, $h_4^{(4)}(b)=5400b^2-120$.

As $h_4^{(4)}(b)=5400b^2-120>0$ for all $b\in(e^{-1},1)$. Then $h_4^{(3)}(b)$ increase monotonically for all $b\in(e^{-1},1)$.

As $h_4^{(3)}(e^{-1})\approx 39.4712>0$, then $h_4^{(3)}(b)>0$ for all $b\in(e^{-1},1)$. Then $h_4''(b)$ increase monotonically for all $b\in(e^{-1},1)$.

As $h_4''(e^{-1})\approx -4.08536<0$ and $h_4''(1)=382>0$, and $h_4''(0.43333)=0$, Then $h_4'(b)$ increase monotonically when $b\in(e^{-1},0.43333)$ and decrease monotonically when $b\in(0.43333,1)$.

For the monotonically increasing part of $h_4'(b)$, since $h_4'(e^{-1})\approx 3.46891>0$, then $h_4'(b)>0$ when $b\in(e^{-1},0.43333)$. Then the monotonically decreasing part of $h_4'(b)$, since $h_4'(1)=70>0$, $h_4'(b)>0$ when $b\in(0.43333,1)$. Overall $h_4'(b)>0$ and thus $h_4(b)$ increase monotonically when $b\in(e^{-1},1)$.

As $h_4(e^{-1})\approx 2.59988>0$, then $h_4(b)>0$ for all $b\in(e^{-1},1)$. Then $g^{(4)}(b)>0$ for all $b\in(e^{-1},1)$. □

Lemma 10. $g^{(3)}(b)=6b(20b^8-10b^6-4b^5-4b^4+9b^3+5b^2+b-1)>0$ *for all $b\in(e^{-1},1)$*

Proof. Let $h_3(b) = 20b^8 - 10b^6 - 4b^5 - 4b^4 + 9b^3 + 5b^2 + b - 1$, then:

$$\begin{aligned}
h_3'(b) =& 160b^7 - 60b^5 - 20b^4 - 16b^3 + 27b^2 + 10b + 1 \\
h_3''(b) =& 1120b^6 - 30b^4 - 80b^3 - 48b^2 + 54b + 10 \\
h_3^{(3)}(b) =& 6720b^5 - 120b^3 - 240b^2 - 96b + 54 \\
h_3^{(4)}(b) =& 33600b^4 - 360b^2 - 480b - 96 \\
h_3^{(5)}(b) =& 33600 \times 4b^3 - 720b - 480 \\
h_3^{(6)}(b) =& 33600 \times 12b^2 - 720
\end{aligned}$$

As $h_3^{(6)}(b) > 0$ for all $b \in (e^{-1}, 1)$, then $h_3^{(5)}(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

As $h_3^{(5)}(e^{-1}) \approx 5946.51 > 0$, then $h_3^{(5)}(b) > 0$ for all $b \in (e^{-1}, 1)$, so $h_3^{(4)}(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

As $h_3^{(4)}(e^{-1}) \approx 294.103 > 0$, then $h_3^{(4)}(b) > 0$ for all $b \in (e^{-1}, 1)$, so $h_3^{(3)}(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

As $h_3^{(3)}(e^{-1}) \approx 25.5077 > 0$, then $h_3^{(3)}(b) > 0$ for all $b \in (e^{-1}, 1)$, so $h_3''(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

As $h_3''(e^{-1}) \approx 21.6132 > 0$, then $h_3''(b) > 0$ for all $b \in (e^{-1}, 1)$, so $h_3'(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

As $h_3'(e^{-1}) \approx 6.91157 > 0$, then $h_3'(b) > 0$ for all $b \in (e^{-1}, 1)$, so $h_3(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

As $h_3(e^{-1}) \approx 0.374347 > 0$, then $h_3(b) > 0$ for all $b \in (e^{-1}, 1)$, so $g_3(b) > 0$ for all $b \in (e^{-1}, 1)$. By the monotonicity of $g^{(3)}(y)$, for all $b < y < 1$, $g^{(3)}(y) > 0$. Therefore $g''(y)$ increases monotonically in the same range. ☐

Lemma 11. *$g''(b) = 2b(15b^9 - 10b^7 - 6b^6 - 6b^5 + 7b^4 + 8b^3 + 4b^2 - 3b - 1) > 0$ for all $b \in (0.552466, 1)$, $g''(b) < 0$ for all $b \in (e^{-1}, 0.552466)$ and $g''(1) > 0$ for all $b \in (e^{-1}, 1)$*

Proof. Let $h_2(b) = 15b^9 - 10b^7 - 6b^6 - 6b^5 + 7b^4 + 8b^3 + 4b^2 - 3b - 1$, then we have:

$$\begin{aligned}
h_2'(b) =& 135b^8 - 70b^6 - 36b^5 - 30b^4 + 28b^3 + 24b^2 + 8b - 3 \\
h_2''(b) =& 1080b^7 - 420b^5 - 180b^4 - 120b^3 + 84b^2 + 48b + 8 \\
h_2^{(3)}(b) =& 1080 \times 7b^6 - 2100b^4 - 720b^3 - 360b^2 + 168b + 48 \\
h_2^{(4)}(b) =& 1080 \times 42b^5 - 8400b^3 - 2160b^2 - 720b + 168 \\
h_2^{(5)}(b) =& 1080 \times 210b^4 - 25200b^2 - 4320b - 720 \\
h_2^{(6)}(b) =& 1080 \times 840b^3 - 50400b - 4320 \\
h_2^{(7)}(b) =& 907200 \times 3b^2 - 50400
\end{aligned}$$

As $h_2^{(7)}(b) > 0$ for all $b \in (e^{-1}, 1)$. Then $h_2^{(6)}(b)$ increase monotonically for all $b \in (e^{-1}, 1)$.

$h_2^{(6)}(e^{-1}) \approx 22305.7 > 0$, then $h_2^{(6)}(b) > 0$ for all $b \in (e^{-1}, 1)$. Then $h_2^{(5)}(b)$ increase monotonically when $b \in (e^{-1}, 1)$.

$h_2^{(5)}(e^{-1}) \approx -1565.7 < 0$, $h_2^{(5)}(1) > 0$ and $h_2^{(5)}(0.41812) = 0$, then $h_2^{(5)}(b) < 0$ when $b \in (e^{-1}, 0.41812)$ and $h_2^{(5)}(b) > 0$ when $b \in (0.41812, 1)$. Then $h_2^{(4)}(b)$ decrease monotonically for all $b \in (e^{-1}, 0.41812)$ and increase monotonically for all $b \in (0.41812, 1)$.

$h_2^{(4)}(e^{-1}) \approx -501.776 < 0$, $h_2^{(4)}(0.41812) \approx -545.02 < 0$, $h_2^{(4)}(1) \approx 34248 > 0$, $h_2^{(4)}(0.5496) = 0$, then $h_2^{(4)}(b) < 0$ for $b \in (e^{-1}, 0.5496)$ and $h_2^{(4)}(b) > 0$ for $b \in (0.5496, 1)$. Then $h_2^{(3)}(b)$ decrease monotonically for $b \in (e^{-1}, 0.5496)$ and increase monotonically for $b \in (0.5496, 1)$.

As $h_2^{(3)}(e^{-1}) \approx 5.51288 > 0$, $h_2^{(3)}(0.5496) \approx -71.1883 < 0$, $h_2^{(3)}(1) \approx 4596 > 0$, $h_2^{(3)}(0.378693) = 0$ $h_2^{(3)}(0.648449) = 0$, then $h_2^{(3)}(b) > 0$ for $b \in (e^{-1}, 0.378693)$ and $b \in (0.648449, 1)$, $h_2^{(3)}(b) < 0$ for $b \in (0.378693, 0.648449)$. Then $h_2''(b)$ increase monotonically for $b \in (e^{-1}, 0.378693)$, decrease monotonically $b \in (0.378693, 0.648449)$, and increase monotonically for $b \in (0.648449, 1)$.

As $h_2''(e^{-1}) > 0, h_2''(0.378693) > 0, h_2''(0.648449) > 0\, h_2''(1) > 0$, then $h_2''(b) > 0$ for all $b \in (e^{-1}, 1)$. Then $h_2'(b)$ increase monotonically for $b \in (e^{-1}, 1)$.

As $h_2'(e^{-1}) > 0$, then $h_2'(b) > 0$ for all $b \in (e^{-1}, 1)$. Then $h_2(b)$ increase monotonically for $b \in (e^{-1}, 1)$.

As $h_2(e^{-1}) \approx -1.09836 < 0$, $h_2(1) = 8 > 0$, $h_2(0.552466) = 0$, then $h_2(b) > 0$ for all $b \in (0.552466, 1)$ and $h_2(b) < 0$ for all $b \in (e^{-1}, 0.552466)$. Then $g''(b) > 0$ for all $b \in (0.552466, 1)$ and $g''(b) < 0$ for all $b \in (e^{-1}, 0.552466)$.

Then we will show that $g''(1) = 30b^6 - 32b^5 - 17b^4 + 6b^3 + 35b^2 - 5b - 1 > 0$ for all $b \in (e^{-1}, 1)$.

Let $h_2(b) = 30b^6 - 32b^5 - 17b^4 + 6b^3 + 35b^2 - 5b - 1$, then

$$\begin{aligned}
h_2'(b) =& 180b^5 - 160b^4 - 68b^3 + 18b^2 + 70b - 5 \\
h_2''(b) =& 900b^4 - 640b^3 - 204b^2 + 36b + 70 \\
h_2^{(3)}(b) =& 3600b^3 - 1920b^2 - 408b + 36 \\
h_2^{(4)}(b) =& 10800b^2 - 3840b - 408 \\
h_2^{(5)}(b) =& 21600b - 3840
\end{aligned}$$

As $h_2^{(5)}(b) = 21600b - 3840 > 0$ for all $b \in (e^{-1}, 1)$. Then $h_2^{(4)}(b)$ increase monotonically for $b \in (e^{-1}, 1)$.

As $h_2^{(4)}(e^{-1}) < 0$, $h_2^{(4)}(1) > 0$ and $h_2^{(4)}(0.44118) = 0$. Then $h_2^{(4)}(b) < 0$ for $b \in (e^{-1}, 0.44118)$ and $h_2^{(4)}(b) > 0$ for $b \in (0.44118, 1)$. Then $h_2^{(3)}(b)$ decrease monotonically $b \in (e^{-1}, 0.44118)$, and increase monotonically for $b \in (0.44118, 1)$.

As $h_2^{(3)}(e^{-1}) < 0$, $h_2^{(3)}(0.44118) < 0$, $h_2^{(3)}(1) > 0$, $h_2^{(3)}(0.67860) = 0$. Then $h_2^{(3)}(b) < 0$ for $b \in (e^{-1}, 0.67860)$ and $h_2^{(3)}(b) > 0$ for $b \in (0.67860, 1)$. Then $h_2^{(2)}(b)$ decrease monotonically $b \in (e^{-1}, 0.67860)$, and increase monotonically for $b \in (0.67860, 1)$.

As $h_2^{(2)}(e^{-1}) > 0$, $h_2^{(2)}(0.67860) < 0$, $h_2^{(2)}(1) > 0$, $h_2^{(2)}(0.57476) = 0$, $h_2^{(2)}(0.76562) = 0$. Then $h_2^{(2)}(b) > 0$ for $b \in (e^{-1}, 0.57476)$, $h_2^{(2)}(b) < 0$ for $b \in (0.57476, 0.76562)$ and $h_2^{(2)}(b) > 0$ for $b \in (0.76562, 1)$. Then $h_2'(b)$ increase monotonically $b \in (e^{-1}, 0.57476)$ and $b \in (0.76562, 1)$, and decrease monotonically for $b \in (0.57476, 0.76562)$.

As $h_2'(e^{-1}) > 0\, h_2'(0.57476) > 0$, $h_2'(0.76562) > 0\, h_2'(1) > 0$, then $h_2'(b) > 0$ for $b \in (e^{-1}, 1)$. Then $h_2(b)$ increase monotonically for $b \in (e^{-1}, 1)$. As $h_2(e^{-1}) > 0$, then $h_2(b) > 0$ for all $b \in (e^{-1}, 1)$. then $g''(1) > 0$ for all $b \in (e^{-1}, 1)$ ☐

Lemma 12. *$g'(b) < 0$ for all $b \in (e^{-1}, 1)$. $g'(1) < 0$ when $b \in (e^{-1}, 0.594173)$. $g'(1) > 0$ when $b \in (0.594173, 1)$.*

Proof.

$$\begin{aligned}g'(b) =&6b^{11}-5b^9-4b^8-4b^7+2b^6+5b^5+5b^4-2b^3-2b^2-1\\ =&(b-1)(b+1)(b^2+1)(6b^7-5b^5-4b^4+2b^3+2b^2+1)\end{aligned}$$

Let $h_1(b)=6b^7-5b^5-4b^4+2b^3+2b^2+1$, then

$$\begin{aligned}h_1'(b) =&42b^6-25b^4-16b^3+6b^2+4b\\ =&b(42b^5-25b^3-16b^2+6b+4)\end{aligned}$$

Let $t(b)=42b^5-25b^3-16b^2+6b+4$, then

$$\begin{aligned}t'(b) =&210b^4-75b^2-32b+6\\ t''(b) =&840b^3-150b-32\\ t^{(3)}(b) =&2520b^2-150\end{aligned}$$

As $t^{(3)}(b)>0$ for all $b\in(e^{-1},1)$. Then $t''(b)$ increases monotonically $b\in(e^{-1},1)$. $t''(e^{-1})\approx-45.3608<0$, $t''(1)=658>0$, $t''(0.50412)=0$, then $t'(b)$ decreases monotonically $b\in(e^{-1},0.50412)$ and increases monotonically $b\in(0.50412,1)$.

As $t'(e^{-1})<0$, $t'(0.50412)<0$, $t'(1)>0$ and $t'(0.71700)=0$. Then $t'(b)<0$ for $b\in(e^{-1},0.71700)$, and $t'(b)>0$ for $b\in(0.71700,1)$. then $t(b)$ decreases monotonically when $b\in(e^{-1},0.71700)$ and increases monotonically when $b\in(0.71700,1)$.

As $t(e^{-1})>0$,$t(0.578329)=0$, $t(0.71700)<0$, $t(0.822)=0$ $t(1)>0$. Then $h_1'(b)<0$ for $b\in(0.578329,0.822)$, and $h_1'(b)>0$ for $b\in(e^{-1},0.578329)$ and $b\in(0.822,1)$. Then $h_1(b)$ decreases monotonically when $b\in(0.578329,0.822)$ and increases monotonically when $b\in(e^{-1},0.578329)$ and $b\in(0.822,1)$.

As $h_1(e^{-1})>0$, $h_1(0.578329)>0$, $h_1(0.822)>0$, $h_1(1)>0$, then $h_1(b)>0$ and $g'(b)<0$ for all $b\in(e^{-1},1)$.

$$\begin{aligned}g'(1) =&6b^6-9b^5-9b^4+6b^3+12b^2-5b-1\\ =&(b-1)(6b^5-3b^4-12b^3-6b^2+6b+1)\\ <&0\end{aligned}$$

Let $h(b)=6b^5-3b^4-12b^3-6b^2+6b+1$, then

$$\begin{aligned}h'(b) =&30b^4-12b^3-36b^2-12b+6\\ h''(b) =&120b^3-36b^2-72b-12=12(5b+1)(2b+1)(b-1)\end{aligned}$$

It is easy to see that $h''(b)<0$ for all $b\in(e^{-1},1)$, then $h'(b)$ decreases monotonically when $b\in(e^{-1},1)$.

As $h'(1)<0$, then $h'(b)<0$ for all $b\in(e^{-1},1)$. Then $h(b)$ decreases monotonically $b\in(e^{-1},1)$. As $h(e^{-1})<0$, $h(1)>0$ and $h(0.594173)=0$, then we can get $h(b)>0$ and $g'(1)<0$ for $b\in(e^{-1},0.594173)$, $h(b)<0$ and $g'(1)>0$ for $b\in(0.594173,1)$. □

Lemma 13. *$g(b)<0$ and $g(1)<0$ for all $b\in(e^{-1},1)$.*

Proof.

$$\begin{aligned}g(b) =&b^{12}-b^{10}-b^9-b^8+b^6+2b^5-b\\ =&b(b-1)^2(b+1)^2(b^2+1)(b^5-b^2-1)\\ =&-b(b-1)^2(b+1)^2(b^2+1)(1+b^2-b^5)\end{aligned}$$

As $1+b^2-b^5>0$ for all $b\in(e^{-1},1)$, $g(b)<0$ for all $b\in(e^{-1},1)$.

$$\begin{aligned}g(1) =&b^6-2b^5-3b^4+4b^3+3b^2-2b-1\\ =&(b-1)^2(b+1)^2(b^2-2b-1)\\ =&-(b-1)^2(b+1)^2(1-b^2+2b)\end{aligned}$$

As $1-b^2+2b>0$ for all $b\in(e^{-1},1)$, $g(1)<0$ when $b\in(e^{-1},1)$. □