DPBloomfilter: Securing Bloom Filters with Differential Privacy

Yekun Ke* Yingyu Liang† Zhizhou Sha‡ Zhenmei Shi§ Zhao Song¶

Abstract

The Bloom filter is a simple yet space-efficient probabilistic data structure that supports membership queries for dramatically large datasets. It is widely utilized and implemented across various industrial scenarios, often handling massive datasets that include sensitive user information necessitating privacy preservation. To address the challenge of maintaining privacy within the Bloom filter, we have developed the DPBloomfilter. This innovation integrates the classical differential privacy mechanism, specifically the Random Response technique, into the Bloom filter, offering robust privacy guarantees under the same running complexity as the standard Bloom filter. Through rigorous simulation experiments, we have demonstrated that our DPBloomfilter algorithm maintains high utility while ensuring privacy protections. To the best of our knowledge, this is the first work to provide differential privacy guarantees for the Bloom filter for membership query problems.

* keyekun0628@gmail.com.

† yingyul@hku.hk. The University of Hong Kong. yliang@cs.wisc.edu. University of Wisconsin-Madison.

‡ shazz20@mails.tsinghua.edu.cn. Tsinghua University.

§ zhmeishi@cs.wisc.edu. University of Wisconsin-Madison.

¶ zsong@adobe.com. Adobe Research.

Contents

1 Introduction

In the current data-rich era, extracting meaningful information from the ever-growing volume of data presents a significant challenge [SS13]. To address this challenge, various data structures have been developed to facilitate the extraction of insights from vast datasets [Cha06], such as the Bloom filter [Blo70], count-min sketch [Cor09], hyperloglog [FFGM07], and so on. Among them, the Bloom filter mainly handles membership queries in big data [Blo70]; count-min sketch handles the frequency of occurrence of a certain type of data in big data [Cor09]; Hyperloglog is used to count the cardinality of a set of data, that is, the number of different elements in this set of data [FFGM07].

In this paper, we focus more on the Bloom filter [Blo70], which is a space-efficient probability data structure that deals with membership queries. Due to its efficient space utilization and low time complexity, it is widely used in various scenarios, especially industry scenarios requiring massive data processing and low-latency response capability. Classical scenarios include database systems and web-cache systems [Gre82, NGP09, ML16, PNB20].

In addition to the scenarios mentioned above, the Bloom filter is also used in various scenarios involving sensitive user data. One usage is the privacy-preserving dataset intersection: When two organizations want to find out what user data they have in common without revealing specific user information, Bloom filters can be used. By converting the respective user datasets into Bloom filters and then performing an intersection operation, common elements can be determined without exposing specific user records [Bud13, JS11]. Another scenario is anonymous login: Bloom filters can store hash values of login credentials. When a user tries to log in, the system can check whether the hash of the credentials may exist in the filter instead of storing the actual password hash [LVD11, BCMP20]. Since the content inserted into the Bloom filter is user-sensitive, preventing attackers from reconstructing user-sensitive information from the released Bloom filter vector is an essential task.

In this work, we consider the differential privacy of the Bloom filter under the membership query scenario. The membership query problem involves storing information about a set of elements S in a space-efficient manner to determine if an element x is a member of S. One example is the membership query application of the Bloom filter in streaming media recommendation [WZW+14], such as Tiktok. That is, the Bloom filter will be used for filtering to prevent users from being recommended duplicate content when using streaming media. The Bloomfilter vector mentioned above will also be released to other businesses, such as advertising, e-commerce, etc. When the Bloomfilter vector is released, the user's privacy information, which videos the user has watched, needs to be well protected.

Thus, we introduce our DPBloomfilter (Algorithm 1) to protect the sensitive user information stored in the Bloomfilter vector, i.e. the m index binary bits based on the hash values generated by k different hash functions. To implement a differential privacy budget, we used the classic random response technique [War65] (Definition 3.4) in differential privacy, which randomly flips some bits to ensure that attackers cannot restore sensitive user data from neighboring datasets (Definition3.2). We theoretically show that our DPBloomfilter achieves (ϵ, δ)-DP guarantee, where the main technique is that we first ensure each bit holds a certain DP guarantee so that we achieve (ϵ, δ)-DP for the entire Bloom filter. Also, we have theoretically proved that our DPBloomfilter has high utility when DP parameters are in a certain regime. Furthermore, our empirical evidence verifies our utility analysis that our DPBloomfilter can procedure membership query services with high accuracy while protecting user data privacy. While providing privacy guarantees, our algorithm preserves the same running complexity as the standard Bloom filter.

Our contribution can be summarized as follows:

- To the best of our knowledge, this is the first work to provide DP for the Bloom filter for membership query problems.
- We have proved from a theoretical perspective that DPBloomfilter can achieve (ϵ, δ)-DP under the random response mechanism while preserving the same running time complexity compared with the standard Bloom filter.
- We have proved from a theoretical perspective that when the DP parameters ϵ and δ are very small, DPBloomfilter can still maintain good utility.
- Our simulation experiments also reflect the same effect as our theoretical results. The two confirm each other.

Roadmap. Our paper is organized as follows. In Section 2, we review related literature. Section 3 presents the preliminary of Bloom Filter and Differential Privacy. In Section 4, we outline the main results of our algorithm. In Section 5, we elaborate the derivations for the closed-form distribution of the random variable W, where N is the $1-\delta$ quantile of W. Section 6 contains the proof of privacy guarantees for DPBloomfilter. Section 7 presents a detailed analysis of utility guarantees for DPBloomfilter. Section 8 restates the analysis results of running time for DPBloomfilter. Section 10 elaborates on the underlying intuitions that informed the design of the DPBloomfilter. In Section 11, we conclude our paper.

2 Related Work

In Section 2.1, we introduce the mechanism and properties, as long as some variants of the Bloom filter. In Section 2.2, we discuss several principle mechanisms used in differential privacy. In Section 2.3, we show the importance of differential privacy in contemporary data mining and recommendation systems.

2.1 Bloom Filter

The Bloom filter is first introduced by [Blo70] and there are many variants of the Bloom filter. One variant is the Cuckoo filter [FAKM14], which "kicks out" the old hash value to another place when a hash conflict occurs. This implementation principle enables it to support the probability data structure of membership queries with deletion operation. Compared with the Standard Bloom filter, it is more suitable for application scenarios with frequent element updates, such as network traffic monitoring [GJH18] and cache system [WYQ+22].

Another variant is the Quotient filter [GFCO18], which differs from the traditional Bloom filter. It implements the heretical storage form of hash value atmosphere quotient and remainder. This approach results in the Quotient filter requiring less storage space and offering faster query speeds than the standard Bloom filter. It is more suitable for membership queries in scenarios with limited resources and high latency requirements [PCD+21, AHA16].

2.2 Differential Privacy

Differential privacy is a technique used to defend against privacy attacks, first proposed by Dwork et al. [DMNS06]. It has become one of the most popular frameworks for ensuring privacy in theoretical analysis and a wide range of application scenarios [LLSY17, YGZ+23, SGP24, LSSS24, LLS+24, LSSZ24, FLL24].

Gaussian mechanism [DMNS06] and Laplace mechanism [DR+14] of DP are widely used techniques to achieve privacy budget. These two mechanisms control the amount of privacy provided by adjusting the variance of the added noise. However, these two mechanisms are very useful when the output is continuous, but they are slightly weak when the output is discrete. However, another classic way to make a data structure private is to add a random responses mechanism [War65], also called a "flip coin". Specifically, some discrete values in the data structure are flipped with a certain probability to achieve privacy [LL23, LL24]. By controlling the probability of flipping, a given privacy budget is achieved.

Over the past decade, numerous works have applied differential privacy to data structures or deep learning models. [KNRS13] applied differential privacy to graph data structure and designed differentially node-private algorithms by projecting input graphs onto bounded-degree graphs, enhancing privacy while maintaining accuracy in realistic network analyses. [WXY+18] introduced an adaptive method for directly collecting frequent terms under local differential privacy by constructing a trie, which can overcome challenges of accuracy and utility compared to existing n-gram approaches. [FI19] focused on applying differential privacy to classical data mining data structures, specifically decision trees, and analyzes the balance between privacy and utility of existing methods. [ZQR+22] demonstrated the integration of differential privacy into linear sketches, ensuring privacy while maintaining high performance in processing sensitive data streams. A related work [AGK12] introduced the BLIP mechanism, which also applies the Random Flip mechanism to the Bloom Filter. Here, we outline the differences between our work and [AGK12] as follows: (*i*) Our proposed DPBloomFilter can satisfy $(\epsilon, \delta)-DP$, while [AGK12] only verified that BLIP mechanism can satisfy ϵ-DP; (*ii*) [AGK12] did not provide theoretical guarantees for the utility of the BLIP mechanism.

2.3 Privacy in Data Mining and Recommendation System

The preservation of privacy is increasingly vital within the realms of data mining and recommendation systems[KMT19].

In data mining, various studies have emerged that concentrate on how to extract knowledge inherited in user behavior data without compromising user privacy. For instance, [WDZ24] introduced a density-based clustering technique incorporating differential privacy. [TCNZ24] delved into the application of local differential privacy (LDP) to forestall privacy violations during the aggregation of user data, in addition to investigating data poisoning attacks on LDP. Moreover, [LZLY23] has managed to maintain both efficiency and availability in mining user behavior features within specific industries while also employing differential privacy. Besides, [SGP24] proposes a novel differentially private GNN that employs a progressive training scheme and aggregation perturbation to enhance privacy while maintaining accuracy.

On the recommendation system front, it has become common practice for streaming media and advertising companies to utilize sensitive user data, including real-time geographic locations, for user recommendations. In response to these privacy concerns, [MM09] first introduced a differential privacy framework tailored for recommendation systems. Besides, [BMG+20] attempts to build a Recommendation with an Attribute Protection (RAP) model, which simultaneously recommends relevant items and counters private-attribute inference attacks. More recently, [XCS24] developed a federated recommendation framework that integrates differential privacy to shield user privacy, reducing the impact of privacy protection on recommendation quality. [HH23] identified a heavy reliance on user data in existing recommendation systems, leaving them susceptible to privacy breaches.

Privacy remains a critical concern in recommendation systems and data mining. This area of study is ripe for further exploration, and addressing these privacy challenges will require a substantial journey ahead.

3 Preliminary

In Section 3.1, we describe the notations we use in this paper. Section 3.2 provides the formal definition of Bloom Filter. Section 3.3 presents the formal definition of Differential Privacy, followed by a discussion on its basic composition in Section 3.4.

3.1 Notations

For any positive integer n, let $[n]$ denote the set $\{1, 2, \cdots, n\}$. We use $\mathbb{E}[]$ to denote the expectation operator and $\Pr[]$ to denote probability. We use $n!$ to denote the factorial of integer n. We use $A_m^n := \frac{m!}{(m-n)!}$ to denote the number of permutation ways to choose n elements from m elements considering the order of selection. We use $\binom{m}{n} := \frac{m!}{n!(m-n!)}$ to denote the number of combination ways to choose n elements from m elements without considering the order of selection. We use $F_X(x)$ to denote the Cumulative Distribution Function (CDF) of a random variable X and use $F_X^{-1}(1-\delta)$ to denote the $1-\delta$ quantile of $F_X(x)$.

3.2 Bloom Filter

A Bloom filter is a space-efficient probabilistic data structure used to test whether an element is a member of the set. Its formal definition is as follows.

Definition 3.1 (Bloom Filter, [Blo70]). *A Bloom filter is used to represent a set $A = \{x_1, x_2, \ldots, x_{|A|}\}$ of $|A|$ elements from a universe $U = [n]$. A Bloom filter consists of a binary array $g \in \{0,1\}^m$ of m bits, which are initially all set to 0, and uses k independent random hash functions $h_1, \ldots, h_k$ with range $\{0, \ldots, m-1\}$. These hash functions map each element in the universe to a random number uniform over the range $\{0, \ldots, m-1\}$ for mathematical convenience. The computation time per execution for all hash functions is $\mathcal{T}_h$. Bloom Filter supports the following operations:*

- INIT(A). *It takes dataset A as input. For each element $x \in A$, the bits $h_i(x)$ of array g are set to 1 for $1 \le i \le k$.*
- QUERY($y \in [n]$). *It takes an element y as input. If all $h_i(y)$ are set to 1, then it outputs a binary answer to indicate that $y \in A$. If not, then it outputs y is not a member of A.*

A Bloom Filter does not have false negative issues but may yield a *false positive* issue, where it suggests that when a query is made to check if an element is in the set but all the positions it maps to are already set to 1 (due to previous insertions of elements of dataset A). Following previous literature [LL23, BCFM98, LK11, LOZ12], we assume a hash function selects each array position with equal probability. Then, the false positive rate of the Bloom Filter defined above can be mathematically approximated by the formula below

$$(1 - e^{-\frac{k|A|}{m}})^k.$$

3.3 Differential Privacy

We begin with introducing the neighboring dataset. We follow the standard definition in the DP literature of "neighboring" for binary data vectors: two datasets are adjacent if they differ in one element. The formal statement is as follows.

Definition 3.2 (Neighboring Dataset, [DMNS06])**.** *$A, A' \in \{0,1\}^n$ are neighboring datasets if they only differ in one element, i.e., $A_i \neq A'_i$ for one $i \in [n]$ and $A_j = A'_j$, for $j \neq i$.*

Differential Privacy (DP) ensures that the output of an algorithm remains statistically similar, under neighboring datasets introduced above, thereby protecting individual privacy. Its formal definition is as follows.

Definition 3.3 (Differential Privacy, [DMNS06])**.** *For a randomized algorithm $M : U \to Range(M)$ and $\epsilon, \delta \geq 0$, if for any two neighboring data u and u', it holds for $\forall Z \subset Range(M)$*

$$\Pr[M(u) \in Z] \leq e^{\epsilon} \Pr[M(u') \in Z] + \delta,$$

then algorithm M is said to satisfy (ϵ, δ)-differentially privacy. If $\delta = 0$, M is called ϵ-differentially private.

Finally, we introduce the formal definition of the random response mechanism.

Definition 3.4 (Random response mechanism)**.** *Let $g \in \{0,1\}^m$ denote the m bit array in the Bloom filter. For any $j \in [m]$, let $\widetilde{g}[j]$ denote the perturbed version of $g[j]$, using the random response mechanism. Namely, for any $j \in [m]$, we have*

$$\Pr[\widetilde{g}[j] = y] = \begin{cases} e^{\epsilon_0}/(e^{\epsilon_0}+1), & y = g[j] \\ 1/(e^{\epsilon_0}+1), & y = 1 - g[j] \end{cases}$$

Let $a = e^{\epsilon_0}/(e^{\epsilon_0}+1), b = 1/(e^{\epsilon_0}+1)$. Since $a/b = e^{\epsilon_0}$, this implies random response can achieve ϵ_0-DP.

3.4 Basic Composition of Differential Privacy

If multiple differential privacy algorithms are involved, a composition rule becomes necessary. This section presents the simplest form of composition, as stated in the following lemma.

Lemma 3.5 (Basic composition, [GKK+23])**.** *Let M_1 be an (ϵ_1, δ_1)-DP algorithm and M_2 be an (ϵ_2, δ_2)-DP algorithm. Then $M(X) = (M_1(X), M_2(M_1(X), X)$ is an $(\epsilon_1+\epsilon_2, \delta_1+\delta_2)$-DP algorithm.*

The basic composition lemma quantifies the total privacy loss across all operations. This is essential for determining whether the overall privacy guarantee remains acceptable.

4 Main Results

In Section 4.1, we will provide the privacy of our algorithm. Then, we will examine the utility implications of our algorithm applying a random response mechanism. In Section 4.2, we introduce the utility guarantees of our algorithm. In Section 4.3, we demonstrate that DPBloomfilter does not import running complexity burden to the standard Bloom filter.

4.1 Privacy for DPBloomfilter

Algorithm 1 illustrates the application of the random response mechanism to the standard Bloom filter, thereby accomplishing differential privacy. In detail, once the Bloom filter is initialized, each bit in the m-bit array is independently toggled with a probability of $\frac{1}{\epsilon_0+1}$. Our algorithm will ensure that modifications to any element within the dataset are protected to a degree, as the DPBloomfilter maintains the privacy of the altered element. Then, we present the Theorem demonstrating that our algorithm is (ϵ, δ)-DP.

Algorithm 1 Differentially Private Bloom Filter

data structure DPBloomFilter ▷ Theorem 4.1, 4.2, 4.3

members
    $[n]$ is the set universe
    k is the number of hash functions
    Let $g \in \{0,1\}^m$.
    Let $h_i : [n] \to [m]$ for each $i \in [k]$
end members

procedure Init($A \subset [n], k \in \mathbb{N}_+, m \in \mathbb{N}_+$) ▷ Lemma 8.1
    Let m denote the length of the filter
    We pick k random hash functions, say they are $h_1, h_2, \cdots, h_k$, for each $i \in [k]$, $h_i : [n] \to [m]$
    Set every entry of g to 0.
    Let $N = F^{-1}(1-\delta)$, and $\epsilon_0 := \epsilon/N$
    for $x \in A$ **do**
        for $i = 1 \to k$ **do**
            Let $j \leftarrow h_i[x]$
            $g[j] \leftarrow 1$
        end for
    end for
    for $j = 1 \to m$ **do**
        $\widetilde{g}[j] \leftarrow g[j]$ with probability $\frac{e^{\epsilon_0}}{e^{\epsilon_0}+1}$
        $\widetilde{g}[j] \leftarrow 1 - g[j]$ with probability $\frac{1}{e^{\epsilon_0}+1}$
    end for
end procedure

procedure Query($y \in [n]$) ▷ Lemma 8.2, Theorem 4.1, Theorem 4.2
    for $i = 1 \to k$ **do**
        Let $j \leftarrow h_i[y]$
        if $\widetilde{g}[j] \neq 1$ **then**
            return false
        end if
    end for
    return true
end procedure

data structure

Theorem 4.1 (Privacy for Query, informal version of Theorem 6.2)**.** *Let $N := F_W^{-1}(1-\delta)$ and $\epsilon_0 = \epsilon/N$. Then, we can show, the output of* Query *procedure of Algorithm 1 achieves (ϵ, δ)-DP.*

Theorem 4.1 shows that our DPBloomfilter in Algorithm 1 is (ϵ, δ)-DP. Our main technique leverages the single-bit random response technique to enhance the privacy properties of the traditional Bloom filter by composition rule (Lemma 3.5).

4.2 Utility for DPBloomfilter

Despite the introduction of privacy-preserving mechanisms, our algorithm still ensures that the utility of the Bloom Filter remains acceptable. This is achieved through careful calibration of the Random Response technique parameters, balancing the need for privacy with the requirement for accurate set membership queries. Here, we present the theorem for the entire utility loss between the output of our algorithm and ground truth.

Theorem 4.2 (Accuracy (compare DPBloom with true-answer) for Query, informal version of Theorem 7.4)**.** *If the following conditions hold*

- *Let* $z \in \{0,1\}$ *denote the true answer for whether* $x \in A$.
- *Let* $\widehat{z} \in \{0,1\}$ *denote the answer for whether* $x \in A$ *output by Bloom Filter.*
- *Let* $\alpha := \Pr[z = 0] \in [0,1]$, $t := e^{\epsilon_0}/(e^{\epsilon_0}+1)$, *and* $\delta_{\mathrm{err}} > 0$.

Then, we can show

$$\Pr[\widetilde{z} = z] \geq \delta_{\mathrm{err}} \cdot \alpha \cdot (1 - t - t^k) + \alpha \cdot t.$$

Theorem 4.2 shows that when most queries are not in A, the above theorem can ensure that the utility of DPBloomfilter has a good guarantee. Namely, in such cases, answers from DPBloomfilter are correct with high probability.

4.3 Running Complexity of DPBloomfilter

Now, we introduce the running complexity for the DPBloomfilter in the following theorem.

Theorem 4.3 (Running complexity of DPBloomfilter)**.** *Let* $\mathcal{T}_h$ *denote the time of evaluation of function* h *at any point. Then, for the DPBloomfilter (Algorithm 1) we have*

- *The running complexity for the initialization procedure is* $O(|A| \cdot k \cdot \mathcal{T}_h + m)$.
- *The running complexity* $O(k \cdot \mathcal{T}_h)$ *for a single query.*

Proof. It can be proved by combining Lemma 8.1 and 8.2. □

Our Theorem 4.3 shows that DPBloomfilter not only addresses the critical need to protect the privacy of elements stored with Bloom filter but also ensures that the data structure's utility remains acceptable, with minimal impact on its computational efficiency. By keeping the running time within the same order of magnitude as the standard Bloom filter, our approach is practical for real-world applications requiring fast and scalable set operations.

5 Proof for $1 - \delta$ Quantile

In this section, we provide the calculation of the probability distribution of random variable $W := \sum_{j=1}^{m} \mathbb{1}\{g[j] \neq g'[j]\}$, which plays an important part in the proof of the privacy guarantee for our algorithm (see Section 6). In Section 5.1, we present the definition of random variables W, Y, Z used in this section. In Section 5.2, we calculate the probability distribution of Y. In Section 5.3, we calculate the probability distribution of Z conditioned on Y. In Section 5.4, we calculate the probability distribution of W.

5.1 Definition

In this section, we present the definitions of random variables which will be used in the section.

Definition 5.1 (Definition of W)**.** *Let $W := \sum_{j=1}^{m} \mathbb{1}\{g[j] \neq g'[j]\}$, where $g \in \{0,1\}^m$ denotes the ground truth values generated by dataset A, and $g' \in \{0,1\}^m$ denotes the ground truth values generated by neighboring dataset A'.*

Definition 5.2 (Definition of Y)**.** *Consider a $x \in [n]$.*

Let $y_1, y_2, \cdots, y_k$ denotes the k hash values generated by the standard Bloom filter (Definition 3.1).

We define Y as the set of distinct values among $y_1, y_2, \cdots, y_k$, where $|Y| \in 1, 2, \cdots, k$.

Definition 5.3 (Definition of Z)**.** *Consider two data $x, x' \in [n]$.*

Let $y_1, y_2, \cdots, y_k$ denotes the k hash values generated by x, and $y'_1, y'_2, \cdots, y'_k$ denotes the k hash values generated by x'.

Follow the Definition 5.2, let Y_x denotes the set of distinct values in $y_1, y_2, \cdots, y_k$, and $Y_{x'}$ denotes the set of distinct values in $y'_1, y'_2, \cdots, y'_k$.

Suppose $|Y_x| = a, |Y_{x'}| = b$, where $a, b \in \{1, 2, \cdots, k\}$

We define Z is the set of distinct values in $Y_x \cup Y_{x'}$, where $|Z| \in \{1, 2, \cdots, 2k\}$

5.2 Distribution of Y

Then we proceed to calculate the probability distribution of Y in this section.

Lemma 5.4 (Distribution of Y)**.** *If the following conditions hold*

- *Let $y_1, y_2, \cdots, y_k$ be defined in Definition 5.2.*
- *Let Y be defined as Definition 5.2.*

Then, we can show, for $y = 1, 2, \cdots, k$,

$$\begin{aligned} & \Pr[|Y| = y] \\ = & \begin{cases} 1/m^{k-1}, & y = 1 \\ \binom{m}{y} \cdot y^k/m^k - \sum_{i=1}^{k-1} \binom{m-i}{y-i} \Pr[Y = i], & y = 2, \cdots, k \end{cases} \end{aligned}$$

Proof. **Step 1.** We consider $Y = 1$ case.

Without any constraints, there are total m^k situations. This is because each hash value can be freely chosen from m positions, and there are k hash values. Therefore, there are total m^k situations.

Then, with constraint $Y = 1$, k hash values must be assigned to the same position. The position can be chosen from a total of m positions. Therefore, in this case, there are m situations.

Combining the above two analysis, we have

$$\begin{aligned} \Pr[Y = 1] &= \frac{m}{m^k} \\ &= \frac{1}{m^{k-1}}. \end{aligned}$$

Step 2. We consider $Y = 2, \cdots, k$ cases.

Similarly, without any constraints, there are total m^k situations.

Since we need $Y = y$, we must choose y from different positions in the total m positions. Therefore, we have $\binom{m}{y}$ term.

Note that in each position, we need at least one hash value. We first compute the number of freely assigning k hash values to the y positions. Then we remove the failure cases.

As there are y positions and k hash values, we have the y^k term for freely assigning k hash values to y positions.

For the failure case, we have $\sum_{i=1}^{k-1} \Pr[Y = i] \cdot \binom{m-i}{y-i}$. The $\binom{m-i}{y-i}$ term is due to repeated counting for each $i \in [k-1]$, where we first fix i positions and then randomly pick the other $y - i$ different positions in the total $m - i$ positions.

Thus, in all, we have the following formula,

$$\Pr[Y = y] = \frac{\binom{m}{y} \cdot y^k}{m^k} - \sum_{i=1}^{k-1} \Pr[Y = i] \cdot \binom{m-i}{y-i}.$$

□

5.3 Distribution of Z conditioned on Y

In this section, we calculate the probability distribution of Z condition on Y.

Lemma 5.5 (Probability of Z conditioned on Y_x and $Y_{x'}$)**.** *If the following conditions hold*

- *Let $Y_x, Y_{x'}, Z$ be defined as Definition 5.3.*
- *Let A_n^m denotes $n!/(n-m)!$.*
- *Let $t := z - \max(a, b)$.*

Then, we can show, for $z = \max(a, b), \cdots, (a + b)$,

$$\Pr[|Z| = z \,|\, |Y_x| = a, |Y_{x'}| = b] = \frac{A_m^a \cdot \binom{b}{t} \cdot A_{m-a}^t \cdot A_a^{b-t}}{A_m^a \cdot A_m^b}.$$

Proof. Since the minimum value of Z is $\max(a, b)$, without loss of generality, we assume $a \geq b$. Then we have $a \leq z \leq (a + b)$.

Recall we have $t = z - \max(a, b) = z - a, t \in \{0, 1, \cdots, b\}$. Then we have

$$\begin{aligned} &\Pr[|Z| = a + t \,|\, |Y_x| = a, |Y_{x'}| = b] \\ = &\frac{A_m^a \cdot \binom{b}{t} \cdot A_{m-a}^t \cdot A_a^{b-t}}{A_m^a \cdot A_m^b}. \end{aligned}$$

We explain why we have the above equation in the following steps.

Step 1. We consider the denominator.

Without any constraints, since $|Y_x| = a$, we need to choose a from different positions in the total m positions. Therefore, we have the A_m^a term in the denominator. Similarly, since $|Y_{x'}| = b$, we have the A_m^b term in the denominator.

Step 2. We consider the numerator.

Firstly, since $|Y_x| = a$, we need to choose a different positions in total m positions. Therefore, we have the A_m^a term in the numerator.

Since Z is defined as Definition 5.3, we can have the following

$$|Y_x \cap Y_{x'}| = a + b - z$$

$$
\begin{aligned}
|Y_{x'}| - |Y_x \cap Y_{x'}| &= z - a \\
&= t
\end{aligned}
$$

Then, we need to choose t values from $Y_{x'}$ to construct $|Y_{x'}| - |Y_x \cap Y_{x'}|$ part. Therefore, we have the $\binom{b}{t}$ term in the numerator.

We also need to choose t different positions in the rest $m - a$ positions for $|Y_{x'}| - |Y_x \cap Y_{x'}|$ part. Hence, we have the A_{m-a}^t term in the numerator.

Lastly, let's consider the $b - t$ part. For this part, we need to choose $b - t$ different positions from a positions. Therefore, we have the A_a^{b-t} term in the numerator.

Combining all analyses together, finally, we have

$$
\Pr[|Z| = z \mid |Y_x| = a, |Y_{x'}| = b] = \frac{A_m^a \cdot \binom{b}{t} \cdot A_{m-a}^t \cdot A_a^{b-t}}{A_m^a \cdot A_m^b}.
$$

$\square$

5.4 Distribution of W

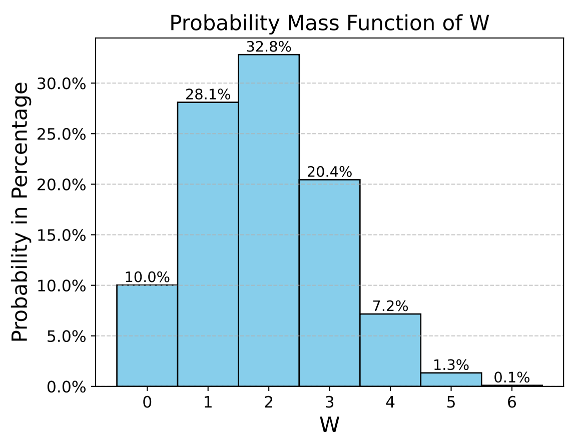

Figure 1: Let $W := |S|$ denote the number of bits in the Bloom filter changed by substituting an element in the inserted set A (Definition 3.2). We achieve ϵ_0-DP for each single bit and (ϵ, δ)-DP for the entire Bloom filter via the random response (Definition 3.4), where $\epsilon_0 = \epsilon/N$. The N is $1-\delta$ quantile of the random variable W. We visualize the distribution of the random variable W (see Lemma 5.6) under the setting described in the experiments section (Section 9). Namely, we have the bit array length in the Bloom filter $m = 2^{19}$, the number of elements inserted into the Bloom filter $|A| = 10^5$, and the number of hash functions $k = 3$. It can be inferred from this visualization that the values of random variable W have good concentration properties, mostly concentrated around its mean.

Finally, we present the calculation of the probability distribution of W in this section.

Lemma 5.6 (Distribution of W)**.** *If the following conditions hold*

- *Let $Y_x, Y_{x'}, Z$ be defined as Definition 5.3.*
- *Let W be defined as Definition 5.1.*
- *Let A_n^m denotes $\frac{n!}{(n-m)!}$.*

- *Let $p_0 := (1-\frac{1}{m})^{(|A|-1)k}$ denotes the proportion of bits which are still 0 in the bit-array.*
- *Let $n_1 := |Y_x \cap Y_{x'}| = a+b-z$ denotes the number of overlap elements in Y_x and $Y_{x'}$.*
- *Let $n_2 := |Y_x \cup Y_{x'}| - |Y_x \cap Y_{x'}| = z-(a+b-z) = 2z-a-b$ denotes the number of exclusive or elements in Y_x and $Y_{x'}$.*

Then, we can show, for $w = 0, \cdots 2k$,

$$\begin{aligned}
&\Pr[W=w] \\
= &\sum_{a=1}^{k}\sum_{b=1}^{k}\sum_{z=1}^{a+b}\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b] \\
&\cdot \Pr[|Z|=z||Y_x|=a,|Y_{x'}|=b] \\
&\cdot \Pr[|Y_x|=a]\cdot\Pr[|Y_{x'}|=b].
\end{aligned}$$

where

$$\begin{aligned}
&\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b] \\
= &\begin{cases} 0, & n_2 < w \\ \binom{n_2}{w}\cdot p_0^w\cdot(1-p_0)^{n_2-w}, & n_2 \geq w \end{cases}
\end{aligned}$$

Proof. By basic probability rules, we have the following equation

$$\begin{aligned}
&\Pr[W=w] \\
= &\sum_{a=1}^{k}\sum_{b=1}^{k}\sum_{z=1}^{a+b}\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b] \\
&\cdot \Pr[|Z|=z||Y_x|=a,|Y_{x'}|=b] \\
&\cdot \Pr[|Y_x|=a,|Y_{x'}|=b] \\
= &\sum_{a=1}^{k}\sum_{b=1}^{k}\sum_{z=1}^{a+b}\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b] \\
&\cdot \Pr[|Z|=z||Y_x|=a,|Y_{x'}|=b] \\
&\cdot \Pr[|Y_x|=a]\cdot\Pr[|Y_{x'}|=b].
\end{aligned}$$

where the first step follows from basic probability rules, the second step follows from Y_x, and $Y_{x'}$ are independent.

We can get the probability of $\Pr[|Y_x|=a]$ and $\Pr[|Y_{x'}|=b$ from Lemma 5.4.

We can get the probability of $\Pr[|Z|=z||Y_x|=a,|Y_{x'}|=b]$ from Lemma 5.5.

Now, let's consider the $\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b]$ term.

Note that only elements in the exclusive-or set may contribute to the final W. Therefore, we have $w \leq n_2$. Namely, when $n_2 < w$, we have $\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b]=0$.

Now, let's calculate $\Pr[W=w||Z|=z,|Y_x|=a,|Y_{x'}|=b]$ under $n_2 \geq w$ condition.

Recall x denotes the element deleted from A, and x' denotes the element added to A for constructing the neighbor dataset A'.

Let $A_{fix} := A - x$ denote the fixed set of elements during the modifications. We have $|A_{fix}| = |A| - 1$.

Consider the following steps:

- We construct a new Bloom filter.
- We insert all elements in A_{fix} in the Bloom filter.
- We define Z_{zero} as the set of positions of bits which are still 0 after the insertion of A_{fix}.

We define Z_{xor} as the exclusive-or set of Y_x and $Y_{x'}$. We have

$$\begin{aligned}
Z_{xor} &= (Y_x \cup Y_{x'}) - (Y_x \cap Y_{x'}), \\
|Z_{xor}| &= |Y_x \cup Y_{x'}| - |Y_x \cap Y_{x'}| \\
&= z - (a + b - z) \\
&= 2z - a - b \\
&= n_2.
\end{aligned}$$

Note that only positions in $Z_{xor} \cap Z_{zero}$ will contribute to W. Namely, we need $|Z_{xor} \cap Z_{zero}| = w$.

We achieve the above condition by selecting w elements in Z_{xor} and let them satisfy the condition of Z_{zero}.

Therefore, we have

$$\begin{aligned}
&\Pr[|Z_{xor} \cap Z_{zero}| = w] \\
= &\binom{n_2}{w} \cdot (1 - \frac{1}{m})^{(|A|-1)kw} \cdot (1 - (1 - \frac{1}{m})^{(|A|-1)k})^{n_2 - w}.
\end{aligned}$$

Combining the above analysis, we have

$$\begin{aligned}
&\Pr[W = w \mid |Z| = z, |Y_x| = a, |Y_{x'}| = b] \\
= &\begin{cases} 0, & n_2 < w \\ \binom{n_2}{w} \cdot p_0^w \cdot (1 - p_0)^{n_2 - w}, & n_2 \geq w \end{cases}.
\end{aligned}$$

□

6 Privacy guarantees for one coordinate

In this section, we provide proof of the privacy guarantees of the DPBloomfilter.

In Section 6.1, we demonstrate the privacy guarantees for single bit of array in Bloom filter.

Then in Section 6.2, we provide the proof of privacy guarantees for our entire algorithm.

6.1 Single bit is private

We first consider the privacy guarantees of single bit of array in Bloom filter.

Lemma 6.1 (Single bit is private). *If the following conditions hold:*

- *Let* $\epsilon_0 \geq 0$.
- *Let* $\widetilde{g}[j] \in \{0, 1\}$ *be the* i*-th element of array output by DPBloomfilter*

Then, we can show that, for all $j \in [m]$, $\widetilde{g}[j]$ *is* ϵ_0*-DP.*

Proof. $\forall j \in [m]$, $g[j]$ is the ground truth value generated by dataset $A \subset [n]$. (An alternative view of g is $g : [m] \to \{0, 1\}$.) Suppose $g[j] = u$, $u \in \{0, 1\}$. For any neighboring dataset $A' \subset [n]$, we denote the ground truth value generated by it as $g'[j]$. Similarly, we can define the $\widetilde{g}'[j]$.

We consider the following two cases to prove $\widetilde{g}[j]$ is ϵ_0-DP, for all $j \in [m]$.

Case 1. Suppose $g'[j] = u$. We know

$$\Pr[\widetilde{g}[j] = u] = \frac{e^{\epsilon_0}}{e^{\epsilon_0} + 1},$$
$$\Pr[\widetilde{g}'[j] = u] = \frac{e^{\epsilon_0}}{e^{\epsilon_0} + 1}.$$

Combining the above two equations, then we obtain

$$\frac{\Pr[\widetilde{g}[j] = u]}{\Pr[\widetilde{g}'[j] = u]} = 1.$$

Similarly, we know

$$\Pr[\widetilde{g}[j] = 1 - u] = \frac{1}{e^{\epsilon_0} + 1},$$
$$\Pr[\widetilde{g}'[j] = 1 - u] = \frac{1}{e^{\epsilon_0} + 1}.$$

Combining the above two equations, then we obtain

$$\frac{\Pr[\widetilde{g}[j] = 1 - u]}{\Pr[\widetilde{g}'[j] = 1 - u]} = 1.$$

Thus, we know for all $v \in \{0, 1\}$,

$$\frac{\Pr[\widetilde{g}[j] = v]}{\Pr[\widetilde{g}'[j] = v]} = 1.$$

Case 2. Suppose $g'[j] \neq u$.

We know

$$\Pr[\widetilde{g}[j] = u] = \frac{e^{\epsilon_0}}{e^{\epsilon_0} + 1},$$
$$\Pr[\widetilde{g}'[j] = u] = \frac{1}{e^{\epsilon_0} + 1}.$$

Combining the above two equations, then we obtain

$$\frac{\Pr[\widetilde{g}[j] = u]}{\Pr[\widetilde{g}'[j] = u]} = e^{\epsilon_0}.$$

Similarly, we know

$$\Pr[\widetilde{g}[j] = 1 - u] = \frac{1}{e^{\epsilon_0} + 1},$$
$$\Pr[\widetilde{g}'[j] = 1 - u] = \frac{e^{\epsilon_0}}{e^{\epsilon_0} + 1}.$$

Combining the above two equations, then we obtain

$$\frac{\Pr[\widetilde{g}[j] = 1-u]}{\Pr[\widetilde{g}'[j] = 1-u]} = e^{-\epsilon_0}.$$

For $v \in \{0,1\}$, we have

$$e^{-\epsilon_0} \leq \frac{\Pr[\widetilde{g}[j] = v]}{\Pr[\widetilde{g}'[j] = v]} \leq e^{\epsilon_0}.$$

Therefore, $\forall j \in [m]$, $\widetilde{g}[j]$ is ϵ_0-DP. $\square$

6.2 Privacy guarantees for DPBloomfilter

Then, we can prove that our entire algorithm is differentially private.

Theorem 6.2 (Privacy for Query, formal version of Lemma 4.1)**.** *If the following conditions hold*

- *Let $N = F_W^{-1}(1-\delta)$ denote the $1-\delta$ quantile of the random variable W (see Definition 5.1).*
- *Let $\epsilon_0 = \epsilon/N$.*

Then, we can show, the output of Query *procedure of Algorithm 1 achieves (ϵ, δ)-DP.*

Proof. Let A and A' are neighboring datasets. Let $g \in \{0,1\}^m$ is the ground truth value generated by dataset A, and $g' \in \{0,1\}^m$ is the ground truth value generated by dataset A'.

We define

$$S := \{j \in [m] \ : \ g[j] \neq g'[j]\}.$$

We further define

$$\overline{S} := [m] \backslash S.$$

We consider two cases, **Case 1** is $j \in \overline{S}$ and **Case 2** is $j \in S$.

Case 1. $j \in \overline{S}$.

We can show that

$$\frac{\Pr[\widetilde{g}[j] = v]}{\Pr[\widetilde{g'}[j] = v]} = 1.$$

holds for $\forall v \in \{0,1\}$.

Case 2. $j \in S$.

We can show that

$$e^{-\epsilon_0} \leq \frac{\Pr[\widetilde{g}[j] = v]}{\Pr[\widetilde{g'}[j] = v]} \leq e^{\epsilon_0}. \tag{1}$$

holds for $\forall v \in \{0,1\}$.

Thus, for any $Z \in \{0,1\}^m$, the absolute privacy loss can be bounded by

$$|\ln \frac{\Pr[\widetilde{g} = Z]}{\Pr[\widetilde{g'} = Z]}| = |\ln \prod_{j \in S} \frac{\Pr[\widetilde{g}[j] = v]}{\Pr[\widetilde{g'}[j] = v]}|$$

$$\leq |S|\epsilon_0$$
$$= |S|\frac{\epsilon}{N}. \tag{2}$$

where the first step follows from each entry of g is independent, the second step follows from Eq. (1), and the last step follows from choice of ϵ_0.

By the definition of N, we know that with probability at least $1-\delta$, $|S| \leq F^{-1}(1-\delta) = N$. Hence, Eq. (2) is upper bounded by ϵ with probability $1-\delta$.

This proves the (ϵ, δ)-DP. □

7 Utility analysis

In this section, we establish the utility guarantees for our algorithm. Initially, we calculate the accuracy for the query of the standard Bloom filter in Section 7.1. We then assess the utility loss caused by introducing the random response technique by comparing the output of the DPBloomfilter with the output of the standard Bloom filter in Section 7.2. Ultimately, we present the assessment of our algorithm's utility in Section 7.3.

We begin by defining the notation we will use in this section.

Definition 7.1. *Let $z \in \{0,1\}$ denote the true answer for whether $x \in A$. Let $\widehat{z} \in \{0,1\}$ denote the answer outputs by* Bloom filter*. Let $\widetilde{z} \in \{0,1\}$ denote the answer output by* DPBloomFilter *(Algorithm 1).*

7.1 Accuracy for query of Standard Bloom Filter

We first present the accuracy of the query of the standard bloom filter, as follows.

Lemma 7.2 (Accuracy for query of Standard Bloom Filter)**.** *If the following conditions hold*

- *Assume that a hash function selects each array position with equal probability.*
- *Let $\widehat{z}$ be defined as Definition 7.1.*
- *Let z be defined as Definition 7.1.*
- *Let $\alpha := \Pr[z = 0]$*

Then, we can show

$$\Pr[\widehat{z} = z] \geq 1 - (1 - e^{-2|A|k/m})^k \cdot \alpha.$$

Further if $m = \Omega(|A|k)$ and $k = \Theta(\log(1/\delta_{err}))$, we have

$$\Pr[\widehat{z} = z] \geq 1 - \delta_{err} \cdot \alpha.$$

Proof. Recall that we have defined Bloom filter in Definition 3.1, it only has false positive error. Therefore, we only need to calculate the following

$$\Pr[\widehat{z} = 1 | z = 0]$$

Recall that $A \subset [n]$ denotes the set of elements inserted into the Bloom filter. And $h_i : [n] \to [m]$ for each $i \in [k]$ denotes k hash functions used in the Bloom filter.

For a query $y \notin A$, we denotes event E_1 happens if the following happens:

$$h_i[y] = 1, \forall i \in [k]$$

Recall that we have defined Bloom filter in Definition 3.1, we have

$$\Pr[\widehat{z} = 1 | z = 0] = \Pr[E_1]. \tag{3}$$

Now, we start calculating $\Pr[E_1]$.

Recall that we assume a hash function selects each array position with equal probability in the lemma statement.

During one inserting operation, the probability of a certain bit is not set to 1 is

$$(1 - \frac{1}{m})^k$$

If we have inserted $|A|$ elements, the probability that a certain bit is still 0 is

$$(1 - \frac{1}{m})^{|A|k} = ((1 - \frac{1}{m})^m)^{|A|k/m} \geq e^{-2|A|k/m}$$

where the last step follows from $(1 - 1/m)^m \geq e^{-2}$ for all $m \geq 2$.

Thus the probability that a certain bit is 1 is

$$1 - (1 - \frac{1}{m})^{|A|k} \leq 1 - e^{-2|A|k/m}.$$

Combining the above fact, we have

$$\begin{aligned} \Pr[E_1] &= (1 - (1 - \frac{1}{m})^{|A|k})^k \\ &\leq (1 - e^{-2|A|k/m})^k. \end{aligned} \tag{4}$$

where the first step follows from the definition of event E_1, the second step follows from $(1-1/m)^m \geq e^{-2}$ for all $m \geq 2$.

Therefore, the accuracy of Bloom filter is

$$\begin{aligned} \Pr[\widehat{z} = z] &= 1 - \Pr[\widehat{z} = 1 | z = 0] \Pr[z = 0] \\ &= 1 - \Pr[E_1]\alpha \\ &\geq 1 - (1 - e^{-2|A|k/m})^k \alpha. \end{aligned}$$

where the first step follows from Bloom filter only has false positive error, the second step follows from the definition of event E_1 and the definition of α, the third step follows from Eq. (4).

□

7.2 Accuracy (compare DPBloomFilter with Standard BloomFilter) for Query

We then assess the accuracy loss caused by the introduction of the random response technique by comparing the outputs of the DPBloomfilter with those of the standard Bloom filter.

Lemma 7.3 (Accuracy (compare DPBloomFilter with Standard BloomFilter) for Query)**.** *If the following conditions hold*

- *Let* $\widehat{z}$ *be defined as Definition 7.1.*
- *Let* $\widetilde{z}$ *be defined as Definition 7.1.*
- *Let* $\alpha := \Pr[z = 0] \in [0, 1]$
- *Let* $t := \frac{e^{\epsilon_0}}{e^{\epsilon_0}+1}$.
- *Let* δ_{err} *be defined as in Lemma 7.2.*

Then, we can show

$$\Pr[\widetilde{z} = \widehat{z}] \geq t \cdot (\alpha - \delta_{\mathrm{err}}).$$

Proof. We denote the query as q.

We define

$$Q := \{j \in [m] \ : \ h_i(q) = j, \ i \in [k]\} \tag{5}$$

We denote $Q[i]$ as the i-th element in Q.

Using basic probability rules, we have

$$\begin{aligned} & \Pr[\widetilde{z} = \widehat{z}] \\ = & \Pr[\widetilde{z} = 1 | \widehat{z} = 1] \Pr[\widehat{z} = 1] \\ + & \Pr[\widetilde{z} = 0 | \widehat{z} = 0] \Pr[\widehat{z} = 0]. \end{aligned}$$

Step 1. Calculate $\Pr[\widetilde{z} = 1 | \widehat{z} = 1]$

We denote event E_2 happens as the following happens:

$$\widetilde{g}[j] = g[j], \forall j \in Q.$$

Recall that we have defined Bloom filter in Definition 3.1, we have

$$\Pr[\widetilde{z} = 1 | \widehat{z} = 1] = \Pr[E_2].$$

Now, we calculate the probability that E_2 happens.

$$\begin{aligned} \Pr[E_2] = & \prod_{i=1}^{k} \Pr[\widetilde{g}[Q[i]] = g[Q[i]]] \\ = & (\frac{e^{\epsilon_0}}{e^{\epsilon_0}+1})^k. \end{aligned}$$

where the first step follows from each entry of g is independent, the second steps follows from the definition of $\widetilde{g}$.

Therefore, we have

$$\Pr[\widetilde{z} = 1 | \widehat{z} = 1] = (\frac{e^{\epsilon_0}}{e^{\epsilon_0}+1})^k. \tag{6}$$

Step 2. Calculate $\Pr[\widetilde{z} = 0 | \widehat{z} = 0]$

Recall we have defined $Q \subset [m]$ in Eq. (5). We further define

$$Z := \{j \in Q \ : \ g[j] = 0\}.$$

We denote $Z[i]$ as the i-th element in Z.
We further define

$$\overline{Q} := Q \backslash Z.$$

By basic probability rules, we have

$$\Pr[\widetilde{z} = 0 | \widehat{z} = 0] = 1 - \Pr[\widetilde{z} = 1 | \widehat{z} = 0].$$

Now, let's calculate $\Pr[\widetilde{z} = 1 | \widehat{z} = 0]$
$[\widetilde{z} = 1 | \widehat{z} = 0]$ happens only if the following conditions hold:

1. All elements in Z flip from 0 to 1.
2. All elements in $\overline{Q}$ remain 1.

Then, we have

$$\begin{aligned}
\Pr[\widetilde{z} = 1 | \widehat{z} = 0] = & \prod_{i=1}^{|Z|} \Pr[\widetilde{g}[Z[i]] = 1] \prod_{i=1}^{|\overline{Q}|} \Pr[\widetilde{g}[\overline{Q}[i]] = 1] \\
= & (\frac{1}{e^{\epsilon_0} + 1})^{|Z|} (\frac{e^{\epsilon_0}}{e^{\epsilon_0} + 1})^{|\overline{Q}|} \\
\leq & (\frac{1}{e^{\epsilon_0} + 1})^{|Z|} \\
\leq & \frac{1}{e^{\epsilon_0} + 1}.
\end{aligned}$$

where the first step follows from the above analysis, the second step follows from the definition of $\widetilde{g}$, the third step follows from $|\overline{Q}| \geq 0$ and $\frac{e^{\epsilon_0}}{e^{\epsilon_0}+1} < 1$, the fourth step follows from $|Z| \geq 1$ and $\frac{1}{e^{\epsilon_0}+1} < 1$.

Therefore, we have

$$\begin{aligned}
\Pr[\widetilde{z} = 0 | \widehat{z} = 0] = & 1 - \Pr[\widetilde{z} = 1 | \widehat{z} = 0] \\
\geq & 1 - \frac{1}{e^{\epsilon_0} + 1} \\
= & \frac{e^{\epsilon_0}}{e^{\epsilon_0} + 1}.
\end{aligned} \tag{7}$$

Let $\widehat{\alpha} := \Pr[\widehat{z} = 0]$, then we have $1 - \widehat{\alpha} = \Pr[\widehat{z} = 1]$. Let $\alpha := \Pr[z = 0]$. Note that $\widehat{\alpha} = \alpha(1 - \delta_{\mathrm{err}})$.

Let $t := \frac{e^{\epsilon_0}}{e^{\epsilon_0}+1}$.

The final accuracy is

$$\begin{aligned}
& \Pr[\widetilde{z} = 0 | \widehat{z} = 0] \cdot \Pr[\widehat{z} = 0] + \Pr[\widetilde{z} = 1 | \widehat{z} = 1] \cdot \Pr[\widehat{z} = 1] \\
= & \Pr[\widetilde{z} = 0 | \widehat{z} = 0] \cdot \widehat{\alpha} + \Pr[\widetilde{z} = 1 | \widehat{z} = 1] \cdot (1 - \widehat{\alpha}) \\
= & \Pr[\widetilde{z} = 0 | \widehat{z} = 0] \cdot \alpha(1 - \delta_{err}) \\
+ & \Pr[\widetilde{z} = 1 | \widehat{z} = 1] \cdot (1 - \alpha + \alpha \cdot \delta_{err})
\end{aligned}$$

$$\geq \frac{e^{\epsilon_0}}{e^{\epsilon_0}+1} \cdot \alpha(1-\delta_{err}) + (\frac{e^{\epsilon_0}}{e^{\epsilon_0}+1})^k \cdot (1-\alpha+\alpha\cdot\delta_{err})$$
$$= t\cdot(\alpha-\alpha\cdot\delta_{err}) + t^k\cdot(1-\alpha+\alpha\cdot\delta_{err})$$
$$\geq t\cdot\alpha\cdot(1-\delta_{err}).$$

where the first step follows from the definition of $\widehat{\alpha}$, the second step follows from $\widehat{\alpha} = \alpha(1-\delta)$, the third step follows from Eq. (6) Eq. (7), the fourth step follows from basic algebra rules, the fifth step follows from $(1-\alpha+\alpha\cdot\delta_{\mathrm{err}}) \geq 0$.

Therefore, the final accuracy is $t\cdot(\alpha-\delta_{err})$. □

7.3 Accuracy (compare DPBloomfilter with true-answer) for Query

Now we can examine the utility guarantees of DPBloomfilter by calculating the error between the ground truth for query and the output of DPBloomfilter.

Theorem 7.4 (Accuracy (compare DPBloomfilter with true-answer) for Query, formal version of Lemma 4.2)**.** *If the following conditions hold*

- *Let $\widehat{z}$ be defined as Definition 7.1.*
- *Let z be defined as Definition 7.1.*
- *Let $\alpha := \Pr[z=0] \in [0,1]$*
- *Let $t := e^{\epsilon_0}/(e^{\epsilon_0}+1)$.*
- *Let δ_{err} be defined as in Lemma 7.2.*

Then, we can show

$$\Pr[\widetilde{z}=z] \geq \alpha(1-t-t^k)\delta_{\mathrm{err}} + \alpha t.$$

Proof. We have

$$\begin{aligned}
& \Pr[\widetilde{z}=z] \\
= \;& \Pr[\widetilde{z}=0|\widehat{z}=0]\Pr[\widehat{z}=0|z=0]\Pr[z=0] \\
+ \;& \Pr[\widetilde{z}=0|\widehat{z}=1]\Pr[\widehat{z}=1|z=0]\Pr[z=0] \\
+ \;& \Pr[\widetilde{z}=1|\widehat{z}=1]\Pr[\widehat{z}=1|z=1]\Pr[z=1] \\
+ \;& \Pr[\widetilde{z}=1|\widehat{z}=0]\Pr[\widehat{z}=0|z=1]\Pr[z=1] \\
\geq \;& t\cdot(1-\Pr[E_1])\cdot\alpha + (1-t^k)\cdot\Pr[E_1]\cdot\alpha + t^k\cdot 1\cdot(1-\alpha) \\
= \;& \alpha(1-t-t^k)\delta_{\mathrm{err}} + \alpha t + t^k(1-\alpha) \\
\geq \;& \alpha(1-t-t^k)\delta_{\mathrm{err}} + \alpha t.
\end{aligned}$$

where the first step from basic probability rules, the secod step follows from Equation 3, Equation 7 and definition of α and t, the third step follows from basic algebra, the fourth step follows from the fact that $t, \alpha \in [0,1]$.

□

To make it easier to understand, we also provide the utility analysis of the Bloom filter under the case of random guess.

Lemma 7.5 (Accuracy for Query under Random Guess)**.** *If the following conditions hold*

- *Let* $\widetilde{z}$ *be defined as Definition 7.1.*
- $\epsilon_0 = 0$*. Namely, each bit in the bit-array of the DP Bloom has* $\frac{1}{2}$ *probability to be set to* 0*, and* $\frac{1}{2}$ *probability to be set to* 1*.*

Then, we can show

$$\Pr[\widetilde{z} = 0] = 1 - \frac{1}{2^k},$$
$$\Pr[\widetilde{z} = 1] = \frac{1}{2^k}.$$

Proof. By the definition of Bloom filter 3.1, the answer $\widetilde{z} = 1$ requires k corresponding positions in the bit-array of the query are all set to 1.

Note that each bit has $\frac{1}{2}$ probability to be set to 1. Therefore, we have

$$\Pr[\widetilde{z} = 1] = \frac{1}{2^k}.$$

Then, we have $\Pr[\widetilde{z} = 0] = 1 - \Pr[\widetilde{z} = 1] = 1 - \frac{1}{2^k}$. □

8 Running Time

In this section, we provide the proof of running time for Algorithm 1. The running time for our algorithm consists of two parts: time for initialization in Section 8.1 and time for query in Section 8.2.

8.1 Running time for initialization

Now we calculate the time of initialization for our algorithm.

Lemma 8.1 (Running time for initialization)**.** *Let* $\mathcal{T}_h$ *denote the time of evaluation of function* h *at any point.*

It takes $O(|A| \cdot k \cdot \mathcal{T}_h + m)$ *time to run the initialization function.*

Proof. **Step 1** Let's consider the initialization of the standard Bloom filter.

A single element x needs $O(k \cdot \mathcal{T}_h)$ time to compute over k hash functions.

There are $|A|$ elements which need to be inserted.

Combining the above two facts, it needs $O(|A| \cdot k \cdot \mathcal{T}_h)$ time to initialise the standard Bloom filter.

Step 2 Let's consider the "Flip each bit" part.

Since there are m bits in the Bloom filter, it takes $O(m)$ time to flip each bit.

Therefore, the initialization function needs $O(|A| \cdot k \cdot \mathcal{T}_h + m)$ time to run.

□

8.2 Running time for query

Then, we proceed to calculate the query time for our algorithm.

Lemma 8.2 (Running time for query)**.** *Let $\mathcal{T}_h$ denote the time of evaluation of function h at any point. It takes $O(k \cdot \mathcal{T}_h)$ time to run each query y in the query function.*

Proof. For each query y, the algorithm needs $O(k \cdot \mathcal{T}_h)$ time to compute the hash values of y over k hash functions.

Therefore, it takes $O(k \cdot \mathcal{T}_h)$ time to run the query function for each query. $\square$

By combing the result of Lemma 8.1 and Lemma 8.2, we can obtain the running of our entire algorithm is $O(|A| \cdot k \cdot \mathcal{T}_h + m)$.

9 Experiments

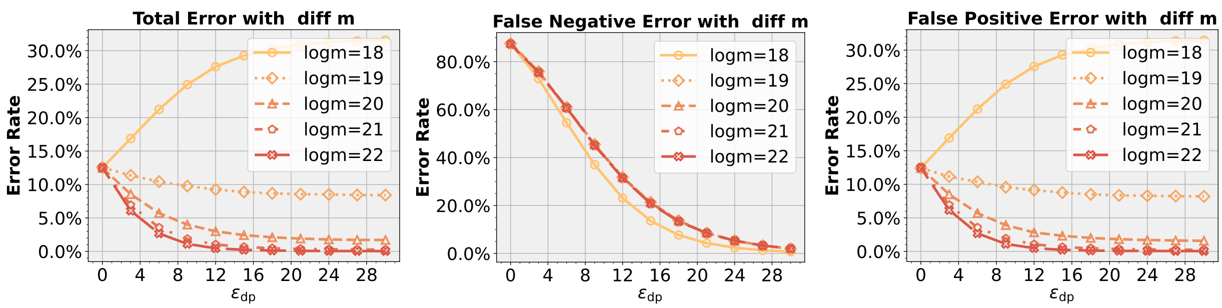

Figure 2: Three kinds of error rates with different bit-array lengths m. We fix the number of inserted elements $|A| = 10^5$, the number of hash functions $k = 3$, and $\delta = 0.01$ in (ϵ, δ)-DP. In the figure, log denotes $\log_2$. **Left:** Total error denotes the case when we randomly choose queries from the universe $[n]$; **Middle:** False negative denotes the case when we randomly choose queries from the set S, which represents the set of elements inserted into the DP Bloom filter; **Right:** False positive denotes the case when we randomly choose queries from the set $\overline{S} = [n] \backslash S$. As m increases, the total error rate and false positive error rate decrease accordingly, while false negative error rate remains constant. As ϵ approaches 0, the DP Bloom filter gets closer to random guessing. In this case, the false positive error rate converges to $\frac{1}{2^k}$, and the false negative error rate converges to $1 - \frac{1}{2^k}$. This is consistent with our result in Lemma 7.5 Our DPBloomFilter achieves practical utility when ϵ is small(e.g. $\epsilon < 10$).

In this section, we introduce the simulation experiments conducted on the DPBloomfilter. In Section 9.1, we introduce the basic setup of our experiments and restate basic definitions of three kinds of error. In Section 9.2, we discuss the results of our experiments, which align with our theoretical analysis.

9.1 Experiments Setup and Basic Notations

Recall that we have the following notations. Let m denote the length of the bit array in the DPBloomfilter. Let $|A|$ denote the number of elements inserted into the DPBloomfilter. Let k denote the number of hash functions used in the DPBloomfilter. Let ϵ, δ denote the differential privacy parameters of the DPBloomfilter. Let N denotes the $1-\delta$ quantile of W (see Definition 5.1),

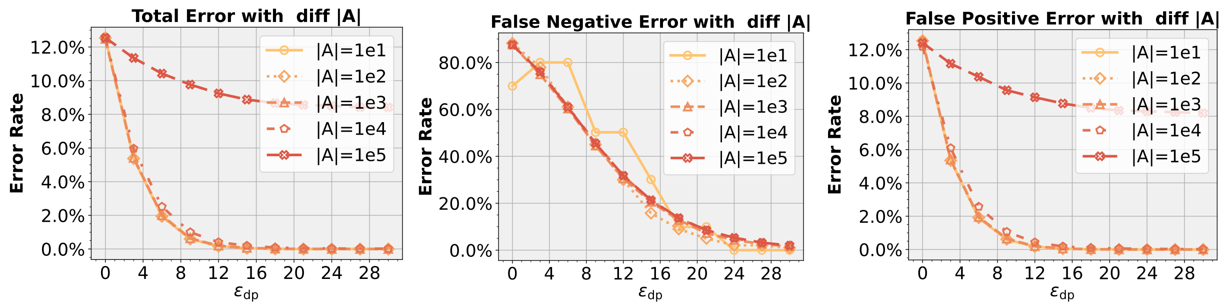

Figure 3: Three kinds of error rates with different numbers of inserted elements $|A|$. We fix the length of bit-array $m = 2^{19}$, the number of hash functions $k = 3$, and $\delta = 0.01$ in (ϵ, δ)-DP. As $|A|$ increases, the Total Error Rate and false positive error rate increase accordingly, while the false negative error rate remains constant.

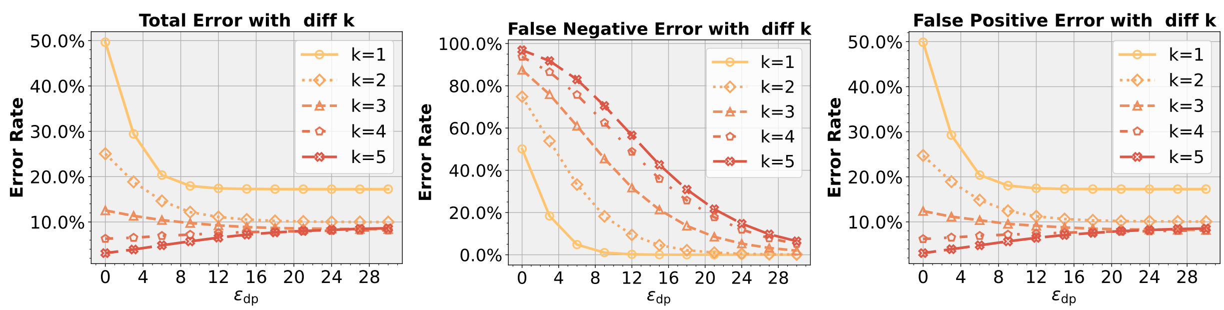

Figure 4: Three kinds of error rates with different numbers of hash function k. We fix the length of bit-array $m = 2^{19}$, the number of inserted elements $|A| = 10^5$, and $\delta = 0.01$ in (ϵ, δ)-DP. As k increases, the Total Error Rate and false positive error rate decrease accordingly, while the false negative error rate increases accordingly.

and the close-form of the distribution of W is shown in Lemma 5.6. Let $\epsilon_0 = \epsilon/N$. By Theorem 4.1, we choose ϵ_0 in this way can guarantee to (ϵ, δ)-DP in the whole algorithm. Unless specified, we adopt $m = 2^{19}, |A| = 10^5, k = 8, n = 2^{63} \approx 10^{19}$ in the following experiments. We choose this n because this n is the biggest integer that can be represented on our server.

Recall that $[n]$ denotes the universe. Let S denote the elements inserted into the DPBloomfilter. Let $\overline{S} = [n] \backslash S$ denote the elements not inserted into the DPBloomfilter. Let $\widetilde{z} \in \{0, 1\}$ denote the answer output by DPBloomfilter.

We report three kinds of error rates in our experiments. They are the following: (1) **total error**, where we randomly choose queries from the universe $[n]$ and report the error rate of our DPBloomfilter; (2) **false positive error**, where we random choose queries from $\overline{S}$. When the DPBloomfilter outputs $\widetilde{z} = 1$, this will cause a false positive error; (3) **false negative error**, where we random choose queries from S. When the DPBloomfilter outputs $\widetilde{z} = 0$, this will cause a false negative error.

9.2 Experiment Results

In this section, we conduct experiments based on the setting mentioned in the previous section. Specifically, we run simulation experiments on different m, $|A|$, and k to demonstrate the utility of our algorithm under differential privacy guarantees.

In Figure 2, we conduct experiments on different m, whereas m increases, the total error rate and false positive error rate decrease accordingly, while the false negative error rate remains constant.

In Figure 3, we also conduct experiments on different $|A|$, whereas $|A|$ increases, the total error rate and false positive error rate increase accordingly. At the same time, the false negative error rate remains constant. This phenomenon is consistent with our theoretical analysis of the utility of DPBloomfilter (Theorem 4.2). Recall that we have $\alpha = \Pr[z = 0]$, denoting the probability of an arbitrary query $q \notin A$. Since $|A|$ increases, α decreases, the utility guarantee in Theorem 4.2, which is consistent with higher error rate in our experiment results.

In Figure 4, we conduct experiments on different k as well, whereas k increases, the total error rate, and false positive error rate decrease, while the false negative error rate increases accordingly.

Note that in Figure 2, Figure 3, and Figure 4, as ϵ approaches 0, the DPBloomfilter gets closer to random guessing. In this case, the false positive error rate converges to $\frac{1}{2^k}$, and the false negative error rate converges to $1 - \frac{1}{2^k}$. This is consistent with our result in Lemma 7.5. Also, as ϵ increases, the three types of error rates in the Bloom filter with differential privacy (DP) approach the error rates observed when DP is not applied. This is consistent with the intuition that when ϵ increases, there is less privacy. Therefore, the performance approaches the performance of a Bloom filter without any privacy guarantees.

10 Discussion

Section 10.1 discusses why the random response mechanism is preferred over Gaussian and Laplace mechanisms for achieving differential privacy. In Section 10.2, we consider the underlying reasons for applying the random response mechanism to both 1 and 0.

10.1 Why Random Response but not Gaussian or Laplace Noise?

As mentioned in Section 2, Gaussian and Laplace noise are two classical mechanisms to achieve differential privacy.

The advantage of the Laplace mechanism is that its distribution is concentrated on its mean. Under the same privacy budget, it will not introduce too much noise like the Gaussian mechanism due to the long-tail nature of its distribution. The advantage of the Gaussian mechanism is that it has good mathematical properties and makes it easy to analyze the utility of private data structures.

However, the above two mechanisms are not as effective as the random response (flip coin) mechanism when dealing with discrete values. Here, we consider the case where the discrete values are integers. Under certain privacy budgets, the noise added by Gaussian and Laplace mechanisms does not reach the threshold of 0.5, resulting in attackers being able to remove the noise through rounding operations easily, and the privacy of the data structure no longer exists.

In our case, each bit of the Bloom filter can only be 1 or 0, which is consistent with the above situation. Hence, our work only considers the random response mechanism instead of classical Gaussian and Laplace mechanisms.

10.2 Why Flip Both 0 and 1?

In our work, we apply random response mechanism to each bit in the Bloom filter, either it is 0 or 1. Although this will lead to a certain probability of false negatives in the Bloom filter, we argue that it is necessary to make the Bloom filter differentially private.

Let's consider what will happen if we don't apply random response mechanism like this. Suppose we only apply random responses to bits that are 1 in the Bloom filter and leave the bits with 0 untouched. Following the notations used in Lemma A, we use $g \in \{0,1\}^m$ to represent the bit array generated by inserting the original dataset into the Bloom filter and $g' \in \{0,1\}^m$ to represent the bit array generated by inserting the neighboring dataset into the Bloom filter. We use $\widetilde{g}$ and $\widetilde{g}'$ to denote their private version, respectively. Without loss of generality, for some $j \in [m]$, we assume $g[j] = 1$ and $g'[j] = 0$. Since we only apply random response mechanism on bits with value 1, then $\Pr[\widetilde{g}'[j] = 1] = 0$. Therefore, we cannot calculate $\Pr[\widetilde{g}[j] = 1] / \Pr[\widetilde{g}'[j] = 1]$, since the denominator is 0. Hence, we cannot have any privacy guarantees under this setting. Similar situations occur when we apply a random response mechanism on bits with value 0. We also cannot prove the differential privacy property of the Bloom filter. Therefore, we have to apply the random response mechanism on bits either with value 0 or 1.

11 Conclusion

In this work, we propose DPBloomfilter, a novel method ensuring the privacy of the Bloom filter via random response. To the best of our knowledge, this is the first work applying the random response mechanism to achieve DP on the membership query task of the Bloom filter. From the privacy side, we have proved that our method achieves (ϵ, δ)-DP with the same running complexity as the standard Bloom filter. From the utility side, we demonstrate from both theoretical and experimental perspectives that the Bloom filter can ensure high utility while ensuring differential privacy.

Acknowledgement

Research is partially supported by the National Science Foundation (NSF) Grants 2023239-DMS, CCF-2046710, and Air Force Grant FA9550-18-1-0166.

References

[AGK12] Mohammad Alaggan, Sébastien Gambs, and Anne-Marie Kermarrec. Blip: non-interactive differentially-private similarity computation on bloom filters. In *Symposium on Self-Stabilizing Systems*, pages 202–216. Springer, 2012.

[AHA16] Mohammad Al-Hisnawi and Mahmood Ahmadi. Deep packet inspection using quotient filter. *IEEE Communications Letters*, 20(11):2217–2220, 2016.

[BCFM98] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

[BCMP20] Davide Berardi, Franco Callegati, Andrea Melis, and Marco Prandini. Password similarity using probabilistic data structures. *Journal of Cybersecurity and Privacy*, 1(1):78–92, 2020.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BMG+20] Ghazaleh Beigi, Ahmadreza Mosallanezhad, Ruocheng Guo, Hamidreza Alvari, Alexander Nou, and Huan Liu. Privacy-aware recommendation with private-attribute protection using adversarial learning. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 34–42, 2020.

[Bud13] Prerna Budhkar. Solving intersection searching problem for spatial data using bloom filters. In *2013 IEEE International Conference on Electronics, Computing and Communication Technologies*, pages 1–5. IEEE, 2013.

[Cha06] Chia-Hui Chang. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering*, 18(10):1411–1428, 2006.

[Cor09] Graham Cormode. Count-min sketch., 2009.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006.

[DR+14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[FAKM14] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.

[FFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.

[FI19] Sam Fletcher and Md Zahidul Islam. Decision tree classification with differential privacy: A survey. *ACM Computing Surveys (CSUR)*, 52(4):1–33, 2019.

[FLL24] Chenglin Fan, Ping Li, and Xiaoyun Li. k-median clustering via metric embedding: towards better initialization with differential privacy. *Advances in Neural Information Processing Systems*, 36, 2024.

[GFCO18] Afton Geil, Martin Farach-Colton, and John D Owens. Quotient filters: Approximate membership queries on the gpu. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 451–462. IEEE, 2018.

[GJH18] Jan Grashöfer, Florian Jacob, and Hannes Hartenstein. Towards application of cuckoo filters in network security monitoring. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 373–377. IEEE, 2018.

[GKK+23] Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, and Kewen Wu. On differentially private counting on trees. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261, page 66. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

[Gre82] Lee L Gremillion. Designing a bloom filter for differential file access. *Communications of the ACM*, 25(9):600–604, 1982.

[HH23] Ming He and Sheng Hu. A normative approach to privacy-preserving recommender systems: Integrating matrix factorization and genetic algorithms. *International Journal of Intelligent Systems*, 2023, 2023.

[JS11] Mark C Jeffrey and J Gregory Steffan. Understanding bloom filter intersection for lazy address-set disambiguation. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 345–354, 2011.

[KMT19] Krishnaram Kenthapadi, Ilya Mironov, and Abhradeep Guha Thakurta. Privacy-preserving data mining in industry. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 840–841, 2019.

[KNRS13] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 457–476. Springer, 2013.

[LK11] Ping Li and Arnd Christian König. Theory and applications of b-bit minwise hashing. *Communications of the ACM*, 54(8):101–109, 2011.

[LL23] Xiaoyun Li and Ping Li. Differentially private one permutation hashing and bin-wise consistent weighted sampling. *arXiv preprint arXiv:2306.07674*, 2023.

[LL24] Ping Li and Xiaoyun Li. Smooth flipping probability for differential private sign random projection methods. *Advances in Neural Information Processing Systems*, 36, 2024.

[LLS+24] Xiaoyu Li, Yingyu Liang, Zhenmei Shi, Zhao Song, and Junwei Yu. Fast john ellipsoid computation with differential privacy optimization. *arXiv preprint arXiv:2408.06395*, 2024.

[LLSY17] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. *Differential privacy: From theory to practice*. Springer, 2017.

[LOZ12] Ping Li, Art Owen, and Cun-Hui Zhang. One permutation hashing for efficient search and learning. *arXiv preprint arXiv:1208.1259*, 2012.

[LSSS24] Yingyu Liang, Zhizhou Sha, Zhenmei Shi, and Zhao Song. Differential privacy mechanisms in neural tangent kernel regression. *arXiv preprint arXiv:2407.13621*, 2024.

[LSSZ24] Yingyu Liang, Zhenmei Shi, Zhao Song, and Yufa Zhou. Differential privacy of cross-attention with provable guarantee. *arXiv preprint arXiv:2407.14717*, 2024.

[LVD11] Rafael P Laufer, Pedro B Velloso, and Otto Carlos MB Duarte. A generalized bloom filter to secure distributed network applications. *Computer Networks*, 55(8):1804–1819, 2011.

[LZLY23] Wenjuan Liang, Wenke Zhang, Songtao Liang, and Caihong Yuan. Privately vertically mining of sequential patterns based on differential privacy with high efficiency and utility. *Scientific Reports*, 13(1):17866, 2023.

[ML16] Ju Hyoung Mun and Hyesook Lim. Cache sharing using a bloom filter in named data networking. In *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, pages 127–128, 2016.

[MM09] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636, 2009.

[NGP09] Marc Najork, Sreenivas Gollapudi, and Rina Panigrahy. Less is more: sampling the neighborhood graph makes salsa better and faster. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 242–251, 2009.

[PCD+21] Prashant Pandey, Alex Conway, Joe Durie, Michael A Bender, Martin Farach-Colton, and Rob Johnson. Vector quotient filters: Overcoming the time/space trade-off in filter design. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1386–1399, 2021.

[PNB20] Ripon Patgiri, Sabuzima Nayak, and Samir Kumar Borgohain. Passdb: A password database with strict privacy protocol using 3d bloom filter. *Information Sciences*, 539:157–176, 2020.

[SGP24] Sina Sajadmanesh and Daniel Gatica-Perez. Progap: Progressive graph neural networks with differential privacy guarantees. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 596–605, 2024.

[SS13] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, pages 42–47. IEEE, 2013.

[TCNZ24] Wei Tong, Haoyu Chen, Jiacheng Niu, and Sheng Zhong. Data poisoning attacks to locally differentially private frequent itemset mining protocols. *arXiv preprint arXiv:2406.19466*, 2024.

[War65] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American statistical association*, 60(309):63–69, 1965.

[WDZ24] Fuyu Wu, Mingjing Du, and Qiang Zhi. Density-based clustering with differential privacy. *Information Sciences*, page 121211, 2024.

[WXY+18] Ning Wang, Xiaokui Xiao, Yin Yang, Ta Duy Hoang, Hyejin Shin, Junbum Shin, and Ge Yu. Privtrie: Effective frequent term discovery under local differential privacy. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 821–832. IEEE, 2018.

[WYQ+22] Yinyin Wang, Yuwang Yang, Xiulin Qiu, Yaqi Ke, and Qingguang Wang. Ccf-lru: hybrid storage cache replacement strategy based on counting cuckoo filter hot-probe method. *Applied Intelligence*, pages 1–15, 2022.

[WZW+14] Shangguang Wang, Zibin Zheng, Zhengping Wu, Michael R Lyu, and Fangchun Yang. Reputation measurement and malicious feedback rating prevention in web service recommendation systems. *IEEE Transactions on Services Computing*, 8(5):755–767, 2014.

[XCS24] Zihang Xu, Chiawei Chu, and Shiyang Song. An effective federated recommendation framework with differential privacy. *Electronics*, 13(8):1589, 2024.

[YGZ+23] Mengmeng Yang, Taolin Guo, Tianqing Zhu, Ivan Tjuawinata, Jun Zhao, and Kwok-Yan Lam. Local differential privacy and its applications: A comprehensive survey. *Computer Standards & Interfaces*, page 103827, 2023.

[ZQR+22] Fuheng Zhao, Dan Qiao, Rachel Redberg, Divyakant Agrawal, Amr El Abbadi, and Yu-Xiang Wang. Differentially private linear sketches: Efficient implementations and applications. *Advances in Neural Information Processing Systems*, 35:12691–12704, 2022.