Differentially Private Sparse Vectors with Low Error, Optimal Space, and Fast Access

Martin Aumüller
maau@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Christian Janos Lebeda
chle@itu.dk
BARC
IT University of Copenhagen
Copenhagen, Denmark

Rasmus Pagh
pagh@di.ku.dk
BARC
University of Copenhagen
Copenhagen, Denmark

ABSTRACT

Representing a sparse histogram, or more generally a sparse vector, is a fundamental task in differential privacy. An ideal solution would use space close to information-theoretical lower bounds, have an error distribution that depends optimally on the desired privacy level, and allow fast random access to entries in the vector. However, existing approaches have only achieved two of these three goals.

In this paper we introduce the Approximate Laplace Projection (ALP) mechanism for approximating k-sparse vectors. This mechanism is shown to simultaneously have information-theoretically optimal space (up to constant factors), fast access to vector entries, and error of the same magnitude as the Laplace-mechanism applied to dense vectors. A key new technique is a *unary* representation of small integers, which is shown to be robust against "randomized response" noise. This representation is combined with hashing, in the spirit of Bloom filters, to obtain a space-efficient, differentially private representation. Our theoretical performance bounds are complemented by simulations which show that the constant factors on the main performance parameters are quite small, suggesting practicality of the technique.

1 INTRODUCTION

One of the fundamental results in differential privacy is that a histogram can be made differentially private by adding noise from the Laplace distribution to each entry of the histogram before it is released [5]. The expected magnitude of the noise on each histogram entry is $O(1/\varepsilon)$, where ε is the privacy parameter, and this is known to be optimal [8]. In fact, there is a sense in which the Laplace mechanism is optimal [10]. However, some histograms of interest are extremely sparse, and cannot be represented in explicit form. Consider, for example, a histogram of the number of HTTP requests to various servers. Already the IPv4 address space has over 4 billion addresses, and the number of unique, valid URLs have long exceeded 10^{12}, so it is clearly not feasible to create a histogram with a (noisy) counter for each possible value.

Korolova, Kenthapadi, Mishra, and Ntoulas [9] showed that it is possible to achieve *approximate* differential privacy with space that depends only on the number of non-zero entries in the histogram. However, for (ε, δ)-differential privacy the upper bound on the expected per-entry error becomes $O\left(\frac{\log(1/\delta)}{\varepsilon}\right)$, which is significantly worse than the Laplace mechanism for small δ. Cormode, Procopiuc, Srivastava, and Tran [4] showed how to achieve pure ε-differential privacy with expected per-entry error bounded by $O\left(\frac{\log(d)}{\varepsilon}\right)$, where d is the dimension of the histogram (number of columns). While both these methods sacrifice accuracy they are very fast, allowing access to entries of the private histogram in constant time. If access time is not of concern, it is possible to combine small space with small per-entry error, as shown by Balcer and Vadhan [2]. They achieve an error distribution that is comparable to the Laplace mechanism (up to constant factors) and space proportional to the sum n of all histogram entries — but the time to access a single entry is $\tilde{O}(n/\varepsilon)$, which is excessive for large data sets.

1.1 Our results

Our contribution is a mechanism that achieves optimal error and space (up to constant factors) with only a small increase in access time. The mechanism works for either approximate or pure differential privacy, with the former providing faster access time. Our main results are summarized in Theorem 1.1.

Theorem 1.1 (Informal Version of Theorems 4.1 and 4.2). *Given privacy parameters $\varepsilon > 0$ and $\delta \geq 0$, there exists an (ε, δ)-differentially private algorithm to represent a k-sparse histogram using $O(k \log(d+u))$ bits with per-entry error matching the Laplace mechanism up to constant factors. The access time is $\log(1/\delta)$ when $\delta > 0$ and $\log(d)$ when $\delta = 0$.*

Here we assume that $k = \Omega(\log(d))$. Otherwise the mechanism has an additional term of $O(\log^2(d))$ or $O(\log(d)\log(1/\delta))$ bits in its space usage for pure and approximate differential privacy, respectively.

1.2 Techniques

We first use a thresholding technique developed in [4, 9] to handle all "large" histogram entries that have at least logarithmic size. To encode the small entries of the histogram, we conceptually switch to a *unary* encoding. In order to pack all unary representations into small space, we use hashing to randomize the position of each bit in the unary representation of a given entry. The access time is logarithmic, and though hash collisions can lead to overestimates, they do not influence the error asymptotically. Finally, privacy is achieved by perturbing each bit in the data structure using randomized response [11]. This is where the unary representation is important: It is redundant enough to allow accurate estimation even when the probability of flipping each bit is a constant bounded away from 1/2. Direct application of randomized response does not give the desired $O(1/\varepsilon)$ dependency, but we solve this issue with an initial scaling step that gives ε-differential privacy when combined with randomized response. Though the discussion above has been phrased in terms of histograms, which makes the comparison to

earlier work easier, our techniques apply more generally to representing sparse real vectors, with privacy for neighboring data sets with bounded ℓ_1-distance.

2 PRELIMINARIES

Problem Setup. In this work, we consider d-dimensional k-sparse vectors of non-negative real values. We say that a vector $x \in \mathbb{R}_+^d$ is k-sparse if it contains at most k non-zero entries. We assume that $k = \Omega(\log(d))$. All entries are bounded from above by a value $u \in \mathbb{R}$, i.e., $\max_{i\in[d]} x_i =: \|x\|_\infty \leq u$. Here $[d]$ is the set of integers $\{1, \ldots, d\}$. We consider the problem of constructing an algorithm $\mathcal{M}$ that releases a differentially private representation of x, i.e., $\tilde{x} := \mathcal{M}(x)$. Note that $\tilde{x}$ does not itself need to be k-sparse.

Utility Measurements. We use two measures for the utility of $\tilde{x} = \mathcal{M}(x)$. We define the *per-entry error* as $|x_i - \tilde{x}_i|$ for any $i \in [d]$. We define the *maximum error* as $\max_{i\in[d]} |x_i - \tilde{x}_i| = \|x - \tilde{x}\|_\infty$. We compare the utility of algorithms using the expected per-entry and maximum error, and compare the tail probabilities of the per-entry error of our algorithm with the Laplace mechanism.

Differential Privacy. In this work, two vectors are neighbors iff their ℓ_1-distance is at most 1. That is for all neighboring vectors $x, x' \in \mathbb{R}_+^d$ we have $\|x - x'\|_1 := \sum_{i\in[d]} |x_i - x'_i| \leq 1$.

Definition 2.1 (Differential privacy [6, Def 2.4]). Given $\varepsilon > 0$ and $\delta \geq 0$, a randomized algorithm $\mathcal{M} \colon \mathbb{R}_+^d \to \mathcal{R}$ is (ε, δ)-differentially private if for all subsets of outputs $S \subseteq \mathcal{R}$ and pairs of k-sparse input vectors $x, x' \in \mathbb{R}_+^d$ such that $\|x - x'\|_1 \leq 1$ it holds that:

$$\Pr[\mathcal{M}(x) \in S] \leq e^\varepsilon \cdot \Pr[\mathcal{M}(x') \in S] + \delta.$$

Probabilistic Tools. Random rounding or stochastic rounding is used for rounding a real value probabilistically based on its fractional part. We define random rounding for any real $r \in \mathbb{R}$ as follows:

$$\text{RandRound}(r) = \begin{cases} \lceil r \rceil & \text{with probability } r - \lfloor r \rfloor \\ \lfloor r \rfloor & \text{with probability } 1 - (r - \lfloor r \rfloor) \end{cases}$$

Randomized response was first introduced by Warner [11]. The purpose of the mechanism is to achieve plausible deniability by changing one's answer to some question with probability p. We define randomized response for a boolean value $b \in \{0, 1\}$ as follows:

$$\text{RandResponse}(b, p) = \begin{cases} 1 - b & \text{with probability } p \\ b & \text{with probability } 1 - p \end{cases}$$

3 THE ALP MECHANISM

In this section, we introduce the Approximate Laplace Projection (ALP) mechanism[1] and give an upper bound on the expected per-entry error. The ALP mechanism consists of two algorithms. The first algorithm constructs a differentially private representation of a k-sparse vector and the second estimates the value of an entry based on its representation.

[1]The name is chosen to indicate that the error distribution is approximately like the Laplace distribution, and that we *project* the sparse vector to a much lower-dimensional representation. It also celebrates the mountains, whose silhouette plays a role in a certain random walk considered in the analysis of the ALP mechanism.

Algorithm 1: ALP-projection

Parameters: $\alpha, \beta, \varepsilon > 0$, and $s \in \mathbb{N}$.
Input: k-sparse vector $x \in \mathbb{R}_+^d$ where $s > 2k$.
Sequence of hash functions from domain $[d]$ to $[s]$, $h = (h_1, \ldots, h_m)$, where $m = \left\lceil \frac{\beta\varepsilon}{\alpha} \right\rceil$.
Output: ε-differentially private representation of x.

(1) Apply random rounding to a scaled version of each non-zero entry of x such that $y_i = \text{RandRound}(\frac{x_i\varepsilon}{\alpha})$.
(2) Construct $z \in \{0, 1\}^{m\times s}$ by hashing the unary representations of y such that:

$$z_{a,b} = \begin{cases} 1, & \exists i : \ b \leq y_i \text{ and } h_b(i) = a \\ 0, & \text{otherwise} \end{cases}$$

(3) Apply randomized response to each bit of z such that $\tilde{z}_{a,b} = \text{RandResponse}(z_{a,b}, \frac{1}{\alpha+2})$.
(4) Release h and $\tilde{z}$.

	$z_{\_,1}$	$z_{\_,2}$	$z_{\_,3}$	$z_{\_,4}$	$z_{\_,5}$	$z_{\_,6}$	$z_{\_,7}$	$z_{\_,8}$
$z_{1,\_}$	0	0	0	0	0	0	0	0
$z_{2,\_}$	1	0	0	0	0	0	0	0
$z_{3,\_}$	0	1	0	1	0	0	0	0
$z_{4,\_}$	0	0	0	0	1	0	0	0
$z_{5,\_}$	0	0	1	0	0	0	0	0

$h_1(i)$	$h_2(i)$	$h_3(i)$	$h_4(i)$	$h_5(i)$	$h_6(i)$	$h_7(i)$	$h_8(i)$
2	3	5	3	4	4	1	2

Figure 1: Embedding with $m = 8$, $s = 5$ and $y_i = 5$. The ith entry is the only non-zero entry.

3.1 Representing a sparse vector

The basic idea is as follows: we apply random rounding to a scaled version of every non-zero entry such that every entry maps to an integer. We then store the unary representation of these integers in a two-dimensional bit-array using a sequence of universal hash functions [3]. We call this bit-array the *embedding*. Lastly, we apply randomized response on the embedding to achieve privacy. The pseudocode of the algorithm is given in Algorithm 1 and we discuss it next.

Figure 1 shows an example of an embedding before applying randomized response. The input is a vector x where the ith entry x_i is the only non-zero value. The result of evaluating i for each hash function is shown in the table at the bottom and the $m = 8$ bits

representing the ith entry in the bit-array have been highlighted. In Step (1) of the algorithm, x_i is scaled by ε/α and randomized rounding is applied to the scaled value. This results in $y_i = 5$. Using the hash functions, we represent this value in unary encoding by setting the first five bits to 1 in Step (4), where the jth bit is selected by evaluating the hash function h_j on i. The final three bits are unaffected by the entry. Finally, we apply randomized response in each cell of the bit-array. The bit-array after applying randomized response is shown in Figure 2. Both the bit-array and the hash functions are the differentially private representation of the input vector x.

The algorithm takes three parameters α, β, and s in addition to the privacy parameter ε. The parameters α and s are adjustable. In general, we set $\alpha = \Theta(1)$ and $s = \Theta(k)$. In the full version of the paper, we discuss the constants hidden in the Theta notation. The parameter β bounds the values stored in the embedding.

The proof that Algorithm 1 satisfies ε-differential privacy is in the full version of our paper. Here we present the intuition behind the proof. First, we analyse the algorithm with privacy parameter $\varepsilon = 1$. Consider the case that only the ith entry in x is changed to construct x' and only a single bit in the embedding is affected by this change. It can be shown that the probability of outputting any value for said bit is within a factor of $e^{|x_i - x'_i|}$. This result is generalized to the case that neighboring databases differ in more than one entry. If two neighboring vectors differ in positions $\mathcal{I}$, the probability of outputting any embedding is within a factor of $\prod_{i \in \mathcal{I}} e^{\|x_i - x'_i\|} = e^{\|x - x'\|_1}$. As such, the algorithm satisfies 1-differentially private as desired. Furthermore, since ε is used to scale the input we have that for any ε and pair of neighboring vectors, the probability of outputting any embedding is within a factor of $e^{\|x - x'\|_1 \cdot \varepsilon} \leq e^{\varepsilon}$.

3.2 Estimating an entry

We now introduce the algorithm to estimate an entry based on the embedding from Algorithm 1. When accessing the ith entry, we estimate the value of y_i and multiply by α/ε to reverse the initial scaling of x_i. As shown above, the ith entry is represented by m bits in the embedding. The estimate of y_i is chosen to maximize a partial sum. If multiple values maximize the sum we chose use their average.

Intuition. The first y_i bits representing the ith entry are set to one before applying noise in Algorithm 1, cf. Figure 1. The last $m - y_i$ bits are zero, except if there are hash collisions. Some bits might be flipped due to randomized response, but we expect the majority of the first y_i bits to be ones and the majority of the remaining $m - y_i$ bits to be zeros. As such the estimate of y_i is based on indices maximizing the difference between ones and zeros prior to the index. The pseudocode for the algorithm is given as Algorithm 2.

Figure 2 shows an example of Algorithm 2. The example is based on the embedding from Figure 1 after adding noise. The plot shows the value of $\sum_{a=1}^{n_i} (2\tilde{z}_{a,(h_a(i))} - 1)$ for all candidate estimates. This sum is maximized at index 3 and 5. This is visualized as the *peaks* in the plot. The estimate is the average of those indices.

In the full paper, we bound the expected per-entry error of Algorithm 2 as well as provide tail bounds. The analysis works by considering a simple random walk that goes up with probability

Algorithm 2: ALP-estimator

Parameters: $\alpha, \varepsilon > 0$.
Input : Embedding $\tilde{z} \in \{0,1\}^{m \times s}$. Sequence of hash functions $h = (h_1, \ldots, h_m)$. Index $i \in [d]$.
Output : Estimate of x_i.

(1) Define the function $f : \{0, \ldots, m\} \to \mathbb{Z}$ as:

$$f(n) = \sum_{a=1}^{n} 2\tilde{z}_{h_a(i),a} - 1$$

(2) Let P be the set of arguments maximizing f. That is,

$$P = \{n \in \{0, \ldots, m\} : f(a) \leq f(n) \text{ for all } a \in \{0, \ldots, m\}\}$$

(3) Let $\tilde{y}_i = \text{average}(P)$

(4) Return $\tilde{y}_i \cdot \frac{\alpha}{\varepsilon}$.

	$\tilde{z}_{\_,1}$	$\tilde{z}_{\_,2}$	$\tilde{z}_{\_,3}$	$\tilde{z}_{\_,4}$	$\tilde{z}_{\_,5}$	$\tilde{z}_{\_,6}$	$\tilde{z}_{\_,7}$	$\tilde{z}_{\_,8}$
$\tilde{z}_{1,\_}$	0	0	0	1	0	0	0	0
$\tilde{z}_{2,\_}$	1	1	0	0	0	1	0	1
$\tilde{z}_{3,\_}$	0	1	0	0	0	0	0	0
$\tilde{z}_{4,\_}$	0	1	0	0	1	0	0	1
$\tilde{z}_{5,\_}$	0	0	1	1	0	0	0	0

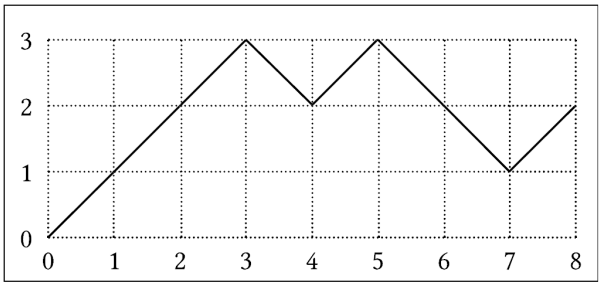

Figure 2: Estimation of ith entry from Figure 1. The partial sum is maximized at indices 3 and 5. The estimate is 4, while the true value was 5.

$p < \frac{1}{2}$ and down with probability $1 - p$ [1]. This random walk models the noise introduced by randomized response and the probability of hash function collisions. The per-entry error can be upper bounded by the position of the last non-negative value in the random walk. Using standard properties of random walks, we show that this position is expected to be constant. Through the rescaling in the last step of the estimation algorithm, this results in an expected error of $O(1/\varepsilon)$. We can also derive tail bounds, and they match the Laplace mechanism up to constant factors for any fixed parameters $\alpha = \Theta(1)$ and $s = \Theta(k)$. However, this only holds for

Algorithm 3: Threshold

Parameters: $\varepsilon, \beta > 0$.
Input : k-sparse vector $x \in \mathbb{R}_+^d$.
Output : ε-differentially private representation of x.

(1) Let $v_i = x_i + \eta_i$ for all $i \in [d]$, where $\eta_i \sim \text{Lap}(1/\varepsilon)$.
(2) Remove entries below β such that:

$$\tilde{v}_i = \begin{cases} v_i, & \text{if } y_i \geq \beta \\ 0, & \text{otherwise} \end{cases}$$

(3) Return $\tilde{v}$.

entries with a true value of at most β. In general, this would mean that we had to set $\beta = u$ if we only were to use the ALP mechanism. The space and access time of the mechanism scale linearly in β. In the next section, we combine our technique with techniques from previous work to lower the value of β.

4 COMBINED DATA STRUCTURE

The ALP mechanism can be combined with techniques from previous work to improve space and access time. As stated above, the ALP mechanism performs well when all entries are bounded by a small value. Some of the algorithms from previous work perform well for large entries. The idea is to combine the mechanism with such an algorithm to construct a composite data structure that performs well for both small and large values.

We use the techniques introduced by Korolova et al. [9] and Cormode et al. [4] for approximate and pure differential privacy, respectively. In this version of the paper, we focus on the pure version based on the algorithm by Cormode et al. The approach is the same for approximate differential privacy. The pseudocode for the algorithm is presented in Algorithm 3.

The algorithms are combined as follows: We represent a k-sparse vector by releasing the output of both Algorithm 1 and 3. By composition [6, Theorem 3.16], the combined representation is ε-differential private if we use privacy parameter $\varepsilon/2$ for both algorithms. When accessing the ith entry, we return the ith entry of $\tilde{v}$ from Algorithm 3 if it is non-zero. Otherwise we run Algorithm 2 on the output of Algorithm 1.

To achieve expected per-entry error $O(1/\varepsilon)$ for any entry, we set $\beta = \frac{2\ln(d/2)}{\varepsilon}$, $\alpha = \Theta(1)$ and $s = \Theta(k)$. Then the output of Algorithm 3 is $O(k)$-sparse with high probability and we can store a $O(k)$-sparse vector using $O(k \log(d+u))$ bits in the standard word RAM model [7]. Furthermore, the embedding of Algorithm 1 can be stored using $O(k \log(d))$ bits and the access time of Algorithm 2 is $O(\log(d))$. The expected maximum error is $O(\log(d)/\varepsilon) + \beta = O(\log(d)/\varepsilon)$.

Theorem 4.1. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, and $\varepsilon > 0$. Then there exists an ε-differentially private algorithm with $O(1/\varepsilon)$ expected per-entry error, $O\left(\frac{\log(d)}{\varepsilon}\right)$ expected maximum error, access time of $O(\log(d))$, and space usage of $O(k \log(d+u))$ with high probability.*

Theorem 4.1 presents our results for pure differentially privacy. When approximate differentially privacy is acceptable, we can instead combine the ALP mechanism with the technique introduced by Korolova et al. to achieve smaller expected maximum error and faster access times. Theorem 4.2 presents our result.

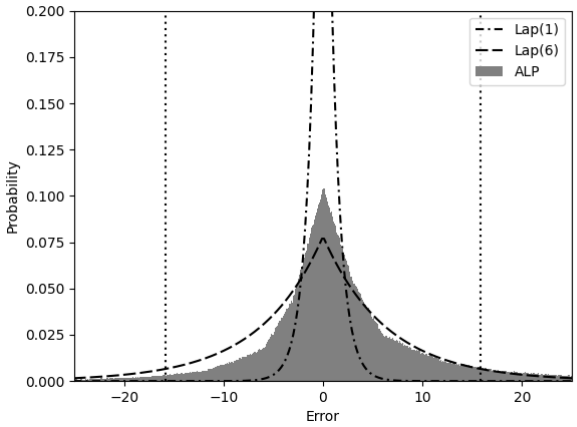

Figure 3: Observed error distribution for parameters $\varepsilon = 1$, $\alpha = 3$ and $\Pr[\text{collisions}] = 0.1$.

Theorem 4.2. *Let $\alpha = \Theta(1)$, $s = \Theta(k)$, and $\varepsilon, \delta > 0$. Then there exist an (ε, δ)-differentially private algorithm with $O(1/\varepsilon)$ expected per-entry error, $O\left(\frac{\log(1/\delta)}{\varepsilon}\right)$ expected maximum error, access time of $O(\log(1/\delta))$, and space usage of $O(k(\log(d+u) + \log(1/\delta)))$.*

5 EXPERIMENTS

In this section, we discuss the per-entry error of Algorithm 2 in practice. We experiment with the impact of adjustable parameters α and s on the expected per-entry error in the full paper. The parameters also determine the constant factors for space usage and access time of the ALP mechanism. The space requirements scale linearly in $\frac{s}{\alpha}$ and the access time scales linearly in $\frac{1}{\alpha}$. As such, the optimal parameter choice differs depending on the use case due to a space, access time, and error trade-off.

Based on our initial experiments, we fixed the parameters to examine the error distribution. We set $\varepsilon = 1$, $\alpha = 3$, and the probability of a hash collision to 0.1. Note that $s = 10k$ implies that the probability of a hash collision is at most 0.1 for any input. We simulated a query 10^6 times. For each simulation we chose the value of x_i uniformly at random in the interval $[0, \ldots, \beta]$.

The error distribution is shown in Figure 3. The mean absolute error of the experiment is 6.4 and the standard deviation is 11. The 90th percentile is 15.78, which is shown in Figure 3 using vertical lines. For comparison, the plots include the Laplace distribution with scale parameters 1 and 6. Note that the Laplace distribution with parameter 1 is optimal for the privacy budget.

The distribution is slightly off-center. This is due to hash collisions. This effect can be reduced by increasing s. This also decrease the expected error, but the space requirements increase. For larger values of s the observed mean absolute error was slightly below 5. Specifically, this was observed for $s \geq 100k$.

The full version of our paper is available at https://arxiv.org/abs/2106.10068.

REFERENCES

[1] Sven Erick Alm. 2002. Simple random walk. (2002).

[2] Victor Balcer and Salil P. Vadhan. 2018. Differential Privacy on Finite Computers. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA (LIPIcs, Vol. 94)*, Anna R. Karlin (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 43:1–43:21. https://doi.org/10.4230/LIPIcs.ITCS.2018.43

[3] J Lawrence Carter and Mark N Wegman. 1979. Universal classes of hash functions. *Journal of computer and system sciences* 18, 2 (1979), 143–154.

[4] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, and Thanh TL Tran. 2012. Differentially private summaries for sparse data. In *Proceedings of the 15th International Conference on Database Theory*. 299–311.

[5] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. 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 (Lecture Notes in Computer Science, Vol. 3876)*, Shai Halevi and Tal Rabin (Eds.). Springer, 265–284. https://doi.org/10.1007/11681878_14

[6] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407. https://doi.org/10.1561/0400000042

[7] Torben Hagerup. 1998. Sorting and Searching on the Word RAM. In *STACS (Lecture Notes in Computer Science, Vol. 1373)*. Springer, 366–398.

[8] Moritz Hardt and Kunal Talwar. 2010. On the geometry of differential privacy. In *Proceedings of ACM symposium on Theory of Computing (STOC)*. 705–714.

[9] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. 2009. Releasing search queries and clicks privately. In *Proceedings of the 18th international conference on World wide web*. 171–180.

[10] Fragkiskos Koufogiannis, Shuo Han, and George J Pappas. 2015. Optimality of the laplace mechanism in differential privacy. *arXiv preprint arXiv:1504.00065* (2015).

[11] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. *J. Amer. Statist. Assoc.* 60, 309 (1965), 63–69.