Private and Efficient Federated Numerical Aggregation

Graham Cormode
Meta
gcormode@meta.com

Igor L. Markov
Meta
imarkov@meta.com

Harish Srinivas
Meta
harishs@meta.com

ABSTRACT

Aggregating data generated locally by smartphones and other edge devices is vital for distributed applications and system-performance monitoring but carries significant risks when data is mishandled. In this work, we develop and deploy numerical aggregation protocols that (*i*) are compatible with several notions of privacy, (*ii*) come with attractive accuracy-privacy tradeoffs when used with *differential privacy*, (*iii*) empirically improve upon prior protocols. Our protocols promote a basic tenet of privacy — *not sharing unnecessary information.* For each private value, at most one bit is used. This supports (*i*) privacy metering that enables privacy controls and (*ii*) worst-case guarantees not covered by differential privacy. We emphasize ease of implementation, compatibility with existing infrastructure, and compelling empirical performance. We report on our experience deploying the method for online aggregation in an industrial context.

1 INTRODUCTION

Smart phones and smart watches, fitness trackers, automotive electronics, building sensors, and other edge devices frequently collect large amounts of private data that would not otherwise be shared, including locations, timestamps, behaviors, personal preferences, as well as data related to financial and medical information. While useful to various tasks such as improving products or online advertising, such data can be stolen and used maliciously, prompting serious concerns and mistrust in technology among the public, government regulators, and some industry participants. Recent laws and regulations limit collection or storage of private data, and operating systems producers (e.g., Apple and Google) are restricting data collection. Learning to work with less private data than before is a challenge for today's technology providers, but also reduces risks. Convincing the public that uses of private data are safe is also challenging [27]. Nevertheless, this work is motivated by the real need of organizations to gather data to monitor the operation of their systems, and receive actionable signals on system health.

Decades of well-intentioned efforts using data de-identification have nevertheless resulted in high profile examples of data leakage [22], showing that it requires considerable work and care to develop reliable privacy-enhancing technologies. More recently, principled technical solutions for improving data privacy are receiving significant attention from academia and industry, including Google (Chrome [15]), Microsoft (Windows [10]), Apple (iOS [9]), Intel (SGX [18]) and others. To limit the impact and potential damage caused by a private datum, it is common to use aggregation whenever possible, and this often suffices. Distributed applications often compute sample mean and variance, the 90th percentile, or a histogram over a sufficiently large sample. For example, federated learning computes sample means for gradient updates. To protect data from leakage before aggregation is complete, Intel offers hardware with Secure Guard Extensions (SGX) [18], which assumes trust in the security of hardware beyond an edge device and in communications with such hardware. Another potential danger is that the aggregated data may reveal some information about individual clients (*in rare cases*). To prevent that, *federated analytics* methods [24] use aggregation protocols with mathematical guarantees (*differential privacy* [14], extended to *federated learning* [9, 21, 23]), where the underlying idea is to add noise to private data before aggregation to provide *plausible deniability* through *randomized response* techniques [31].

Under differential privacy, any contribution of any client is compatible with any private values for that client, but some contributions are more likely. The strength of differential privacy is captured by the parameter $\epsilon > 0$ which (via $\exp(\epsilon)$) bounds the likelihood of truthfully reporting about the correct data — the lower the ϵ, the stricter the privacy requirement. *Local differential privacy* extends these guarantees to each client locally regardless of third-party servers [30], so that aggregators do not see original data.

We describe progress in several directions: (*i*) improving efficiency of numerical aggregation, with and without differential privacy (DP), (*ii*) providing easy-to-understand worst-case guarantees on disclosure of private data that add to probabilistic guarantees from DP, (*iii*) deploying the scheme via a broad-use implementation built upon company infrastructure. While mathematically rigorous, DP techniques *alone* do not currently appear compelling to the media [16], the public, civic groups [25], government regulators, the industry [19] and even the research community [11][30, Section 7]. In particular, differential privacy *alone* does not provide common-sense worst-case privacy guarantees that consumers and product managers tend to assume — that an individual's data are safe from unauthorized parties (such parties are not considered in the DP formalism). To this end, we describe how we implemented numerical aggregation in an end-to-end industry infrastructure to provide *both* intuitive privacy via disclosure limitation and formal DP guarantees for various metric and system health monitoring tasks. We share insights from our deployment experience at Meta to inform other privacy implementations.

1.1 Our approach and contributions

Our work seeks simpler, more reliable, and more efficient techniques for numerical aggregation compatible with or extending prior solutions, suitable for practical deployment within federated systems [4]. In doing so, we do not introduce novel differentially private mechanisms, but rather we provide a DP guarantee for our techniques by leveraging off-the-shelf mechanisms as needed. Our approach naturally accommodates asynchronous updates, whereas secure aggregation can require batching a sufficient number of updates to provide privacy.

Bit-efficient numerical aggregation. Our technical contributions include the introduction, analysis and evaluation of more efficient techniques for the computational estimation of means, variances, and related quantities. These have applications to measuring statistics directly (e.g., to evaluate an experiment), and to training ML models. We approximate real numbers with fixed-point (or integer) representations, expand them in binary, select some of the resulting bits, and postprocess those bits before communicating them. We refer to our techniques as "bit-pushing". For b-bit integers, bit-pushing samples at most one bit per value, providing worst-case privacy guarantees that many end-users can understand. Intuitively, ensuring comparable accuracy would require drawing samples from b times more clients. However, our optimization of bit-sampling frequencies limits the required increase of population size by an empirically-low constant factor. Sampling individual binary digits and communicating them in isolation may seem natural in retrospect, but recent literature in the field pursues other, more complicated approaches and does not report industry deployments.

Several considerations are both novel and important to the correctness and efficiency of our approach to make it suitable for deployment: (1) the bits we extract from the numbers to be aggregated form a *linear decomposition* of such numbers in each bit[1]; (2) communicated bits have distinct weights associated with them (in the linear decomposition); (3) the sets of numbers that share a specified value of some bit do not usually form ranges, but do overlap (Section 2 surveys prior work). Additional considerations are helpful to simplify analysis, streamline implementation or improve performance: we sample bits using a custom-optimized frequency distribution; sampling frequencies can be adjusted based on early sampling results—in particular, unused bits (with estimated mean 0) do not need to be sampled; where genuine random sampling is unnecessary, we replace it with quasi-Monte Carlo (QMC) methods, which reduces scope for poisoning attacks.

Validation and deployment. We describe the validation work done to understand the performance of bit-pushing prior to deployment, and report our additional experiences in online deployment. We show empirically that gathering statistics using the new binary-expansion approach offers significant advantages over established approaches as well as recently proposed techniques. Our formal analysis shows that the error decreases quickly, proportionally to $1/\sqrt{n}$, where n is the number of client bit reports (formalized in Lemma 3.1). In practice, we see that gathering reports from a few thousand users is sufficient to achieve a normalized RMSE of around 3% for a 10-bit quantity, and ten thousand reports ensure that the error level is comfortably below 1%. We observe that the efficiency of other approaches often relies on knowing tight bounds on the range in which the values fall. We relax this assumption and allow our approach to adapt to the distribution of values observed in practice. Last, we observe that mean estimation is not so meaningful for quantities with high skew: estimates based on sampling will inevitably be high-variance or biased, no matter how the samples are aggregated. Instead, our method can report an upper bound on the aggregated samples, and flag when this bound changes significantly over time, indicating a heavy-tail and/or non-stationary distribution. With this understanding, we have deployed bit-pushing online to gather performance statistics at scale in our FA stack.

Privacy metering. In addition to improved bit-efficiency, bit-pushing supports novel privacy controls where private data is metered not at the value level (such as an integer representing someone's current longitude), but at the bit level. Rather than transmit an entire private value with noise added, our aggregation protocols only transmit a single private bit (and limit subsequent bits per value and per client)[2]. To this end, we show how to perform many types of aggregation by combining random bits from different edge devices. At a time when many federated analytics stacks are under active development, our work suggests that limits on transmitting private information can be surfaced as controls at the level of mobile platforms and, perhaps, help the public improve trust in technology. Deploying privacy metering is beyond the scope of this work.

2 PRIOR WORK

ϵ-Local Differential Privacy (ϵ-LDP or just LDP) requires that for any possible output value O emitted by the client, the probability of producing O for a different input varies by at most a factor of $\exp(\epsilon)$. Many LDP mechanisms use randomized response, wherein a binary value is reported accurately with probability $p \geq \frac{1}{2}$, else its complement is sent. Duchi *et al.* combine randomized response with randomized rounding in an early work on LDP [13]. Assuming that input value x is pre-scaled so that $0 \leq x \leq 1$, we can treat x as a probability, and represent it with 1 with probability x, else with 0 (*randomized rounding*), then apply randomized response. Gathering many such noisy binary reports from clients, we can compute an unbiased estimate of the population mean, with variance bounded as a function of the privacy parameter ϵ. Similar ideas have been deployed for Windows app usage-data collection [10]. Later modifications by Wang *et al.* choose values closer to the input with higher probability than those that are further away, giving a "piecewise" mechanism [28].

Communication-efficient numerical aggregation (regardless of privacy) is illustrated by theoretical work on low-bandwidth sensor networks [20] with "the constraint that the communication from each sensor to the fusion center must be a one-bit message." Despite different objectives and resource metrics, basic analysis by Luo [20] concords with ours: in the absence of further information about the data distribution, sending binary digits with exponentially decreasing probabilities is an optimal strategy. More recent interest in communication-efficient transmission of values (also not tied to privacy) has been motivated by machine-learning applications. Dealing with multi-dimensional data, distributed ML applications may be able to leverage bit-level efficiency to reach packet efficiency. Ben-Basat *et al.* analyze leading approaches for estimating a real value using a single bit sent from a client to a server, as a function of the amount of shared randomness between the two parties [3]. In our setting, the relevant point of comparison is given by *subtractive dithering*.[3] For an input value $0 \leq x \leq 1$, each client samples $h \in U[0, 1]$, and sends $b = 1$ if $x \geq h$. The server receives b and h, and estimates $\hat{x} = b + h - 0.5$. This offers some privacy when $h \approx \frac{1}{2}$, but when h is chosen to be close to 0 or 1, we can learn if the value of x is close to h. The variance for each estimate (between 0 and 1) is bounded by a constant. In order to use subtractive dithering as a baseline for comparisons to our work, we apply randomized response to the input-dependent output b to get an LDP guarantee.

[1] Note though that *signed* binary expansions are not linear in the sign bit.

[2] Reducing the private data communication to a single bit provides an intuitive level of privacy that can be appreciated by non-expert users. However, a formal privacy guarantee is often required *in addition*. Our goal in this work is to engineer simple procedures that provide an intuitive notion of privacy augmented by a formal guarantee.

[3] When we evaluated in our setting several approaches that were described in [3], subtractive dithering was a clear frontrunner.

Algorithm 1: Basic bit-pushing algorithm

```
Input  : No. of bits $b$, bit weights $p$, no. of clients $n$
Output : (Result $r$, mean of bits $m$, sum of bits $s$)
Initialize result $r = 0$
for $j = 0$ to $b - 1$ in parallel do
    Contact $c[j] = p[j] \cdot n$ clients to request bit $j$
    Gather weighted sum of bits as $s[j]$
    Compute bit means $m[j] = s[j]/c[j]$
    $r \leftarrow r + (2^j \cdot m[j])$
return $(r, m, s)$
```

The need for adaptive protocols. The methods above assume inputs in the range $[0, 1]$ or, equivalently, in some range $[L, H]$ so they can be mapped to $[0, 1]$ via $f(x) = \frac{x-L}{H-L}$. Assuming loose bounds on the input values has a negative impact on accuracy: while methods that are optimal for $[0, 1]$ can be applied to $[L, H]$, the variance of their estimates scales with $(H - L)^2$ [3]. More promising are protocols that adapt to the data distribution and "zoom in" on the range where the data truly lies. We implement such adaptation in our protocols using a small number of rounds and show (*i*) sharper analytical bounds for variance, backed by (*ii*) reduced variance in simulations. When combined with the intuitive nature of the bit-level privacy metering and ease of achieving a formal ϵ-LDP guarantee, bit-pushing becomes an attractive option for mean estimation and related tasks. Our approach to localizing the range of the data could also be combined with other methods to estimate the mean within the resulting range. The advantage of the bit-pushing approach to find the data range is that it operates with only a single-round of interaction, rather than multiple rounds required by binary search.

3 THE BIT-PUSHING APPROACH

We outline bit-pushing for mean estimation, and formally analyze its properties. Bit-pushing can be used as a subroutine in many applications including federated learning, and can be extended to aggregate some nonlinear quantities.

3.1 Basic bit-pushing algorithm

Assume that each client i out of n owns a private value x_i: we work with b-bit integer and fixed-point values. In the narrative below, we first assume non-negative integers, but this is not a limitation of the approach. Our goal is to estimate the mean $\bar{x} = \sum_{i=1}^{n}(x_i/n)$. We write $x^{(j)}$ to denote the j'th bit in the binary representation of x, and $\bar{x}^{(j)}$ to denote the j'th bit of the mean, $\bar{x}$. In the basic form of bit-pushing, each client selects bit j with probability p_j, and sends the value of their input at this bit location, as the pair $\langle x_i^{(j_i)}, j_i \rangle$. An alternative is where the server randomly selects a p_j fraction of clients to report back on bit j. This reduces variance in the number of reports of each bit and removes the need for the server to specify a sampling distribution to clients. Unless stated otherwise, we adopt this quasi-Monte Carlo sampling method by default. Algorithm 1 gives pseucocode for the core functionality of bit-pushing, given a probability vector p of weights to sample bits with. The full proofs of all claims and accompanying empirical sensitivity analysis are presented in an extended technical report [8].

Lemma 3.1. *The basic bit-pushing protocol provides an estimate that is unbiased and has variance equal to* $\frac{1}{n}\sum_{j=0}^{b-1}\frac{4^j \bar{x}^{(j)}(1-\bar{x}^{(j)})}{p_j}$.

Proof. Let $X^{(j)}$ denote the distribution of the j'th bit value. We can assume that each $X^{(j)}$ follows a Bernoulli distribution with parameter $\mathbb{E}[X^{(j)}]$. Assuming the quasi-Monte Carlo case, where bit j is reported on by exactly np_j clients, our estimate $\hat{X}^{(j)}$ is the mean of these np_j reports. Clearly $\mathbb{E}[\hat{X}^{(j)}] = \mathbb{E}[X^{(j)}]$, which, by linearity of expectation, is $\bar{x}^{(j)}$. Our estimate X is the sum of these bit means, weighted by 2^j, so $X = \sum_{j=0}^{b-1} 2^j \hat{X}^{(j)}$ (applying linear decomposition), and by definition,

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{j=0}^{b-1} 2^j X^{(j)}\right] = \sum_{j=0}^{b-1} 2^j \bar{x}^{(j)} = \bar{x}. \quad (1)$$

For bit j, each report on this bit is assigned weight 2^j. The corresponding contribution to the variance is $\mathbb{V}[2^j X^{(j)}] = 4^j \bar{x}^{(j)}(1-\bar{x}^{(j)})$. Averaged over the np_j reports, the contribution to the variance from the estimate of bit j is $4^j \bar{x}^{(j)}(1 - \bar{x}^{(j)})/(np_j)$, so the overall variance of the estimator is

$$\mathbb{V}[X] = \sum_{j=0}^{b-1} \frac{4^j}{np_j} \bar{x}^{(j)}(1 - \bar{x}^{(j)}) := \frac{1}{n}\sum_{j=0}^{b-1} \frac{\beta_j}{p_j} \quad (2)$$

□

Corollary 3.2. *If each client sends b_{send} bits, the variance decreases to* $\frac{1}{nb_{\text{send}}}\sum_{j=0}^{b-1}\frac{4^j \bar{x}^{(j)}(1-\bar{x}^{(j)})}{p_j}$

This result relies on the fact that the mean is a linear function of the values at each bit location. Clearly, the quality of this bound will depend on the choice of the sampling probabilities p_j, so we consider different choices for setting the p_j values. We note that it is likely that individual bits will be correlated – for example, if the mean is close to a power of 2, then it is much more likely that the top-two most significant bits of the input will be 01 or 10 than 11 or 00. This does not affect our variance bounds. If the number of bits sent by the client, b_{send}, is 1, then such correlations do not impact the protocol, since each sampled input value is independent of the others. Meanwhile, if $b_{\text{send}} > 1$, then bit correlations will have negative covariance, and so reduce the variance of our estimates further. In what follows, we focus on the case $b_{\text{send}} = 1$.

Uniform sampling probabilities. The simplest setting is to pick $p_j = 1/b_{\max}$, i.e., each bit is uniformly likely to be picked. In this case, our bound on the variance becomes $\frac{b_{\max}}{n}\sum_{j=0}^{b_{\max}-1}\bar{x}^{(j)}(1-\bar{x}^{(j)})4^j \leq \frac{b_{\max}}{3n}4^{b_{\max}}$ (since $x(1-x) \leq \frac{1}{4}$ for $0 \leq x \leq 1$). Commonly, we may expect that the mean value is proportional to $2^{b_{\max}}$ (i.e., when $\bar{x}^{(b_{\max})}$ is a constant). Then we can write the variance as proportional to $C_{b_{\max}}\bar{x}^2/n$, and the expected absolute error is proportional to $\sqrt{b_{\max}} \cdot \bar{x}/\sqrt{n}$. However, this is a suboptimal choice, and we can do strictly better as discussed below. Increasing b_{send} from 1 towards $b_{\max}$ removes the dependence on $b_{\max}$ in the variance, although when $b_{\text{send}} = b_{\max}$, the algorithm is equivalent to sending the client's entire input value.

Weighted sampling probabilities. It seems intuitive that higher-order bits should have greater probability of being sampled, since they contribute more highly to the computation. Some natural choices are $p_j \propto 2^j$, or more generally, $p_j \propto c^j = 2^{\alpha j}$ for some c or α. We see next that this is indeed a principled choice.

Optimizing sampling probabilities. The optimal bit sampling probabilities are determined by the variance of each bit.

Lemma 3.3. *The variance of the bit-pushing estimator is minimized by picking* $p_j = \sum_{j=0}^{b-1}\frac{\sqrt{\beta_j}}{A}$*, where* $\beta_j = 4^j \bar{x}^{(j)}(1 - \bar{x}^{(j)})$*, and* $A = \sum_{j=0}^{b-1}\sqrt{\beta_j}$.

Proof. For a fixed budget of bit samples, we seek to minimize $\mathbb{V}[X] = \frac{1}{n}\sum_j \frac{\beta_j}{p_j}$ with $\beta_j, p_j > 0$. To optimize variance in terms of p_j such that $\sum_j p_j = 1$, we perform unconstrained optimization of

$$f(p_1, \ldots, p_{k-1}) = \frac{\beta_k}{1 - \sum_{j=0}^{k-1} p_j} + \sum_{j=0}^{k-1} \frac{\beta_j}{p_j} \qquad (3)$$

Seeking a local extremum inside the probability simplex, we find

$$\forall i = 1..k-1, \ \frac{\partial f(p_1, \ldots, p_k)}{\partial p_i} = \frac{\beta_k}{(1 - \sum_{j=0}^{k-1} p_j)^2} - \frac{\beta_i}{p_i^2} = 0 \qquad (4)$$

$$\text{Therefore} \quad \forall i, l \ \ \frac{\beta_i}{p_i^2} = \frac{\beta_l}{p_l^2} \quad \Rightarrow \quad p_i / p_l = \sqrt{\beta_i / \beta_l} \qquad (5)$$

and this extends to $l = k$ via renumbering. Therefore, $p_j = \sqrt{\beta_j}\frac{p_k}{\sqrt{\beta_k}}$, and to find p_j, we can just L_1-normalize the vector of $\sqrt{\beta_j}$. To confirm this unique critical point as the global minimum, we compute

$$\forall i, j \ \frac{\partial \mathbb{V}}{\partial p_i \partial p_j} = \begin{cases} 0, & \text{for } i \neq j \\ 2\beta_j / p_j^3 n & \text{for } i = j \end{cases} \qquad (6)$$

Given that $p_j, \beta_j > 0 \ \forall j$, the Hessian is positive semidefinite. □

From Lemma 3.1 and our independence assumption, we have $\beta_j = \bar{x}^{(j)}(1 - \bar{x}^{(j)})4^j$, where $\bar{x}^{(j)}$ denotes the mean value at the j'th bit index. If we simply bound the contribution of $\bar{x}^{(j)}(1 - \bar{x}^{(j)})$ values by $\frac{1}{4}$, so that $\beta_j = \frac{1}{4}4^j$, this leads us to set $p_j = 2^j/(2^b - 1)$, and so we obtain

$$\mathbb{V}[X] \leq \frac{1}{n}\sum_j (2^b - 1)\bar{x}^{(j)}(1 - \bar{x}^{(j)})2^j < \bar{x}(2^{b-2})/n. \qquad (7)$$

If the inputs make use of most input bits, then $\bar{x}$ is reasonably large, i.e., $\bar{x} \propto 2^b$, and so $\mathbb{V}[X] \propto \bar{x}^2/n$. Hence, the expected absolute error will be of magnitude $\bar{x}/\sqrt{n}$. Sending $b_{\text{send}} > 1$ bits per client would further reduce this absolute error by a factor of $1/\sqrt{b_{\text{send}}}$.

Local vs. central randomness. On the surface, it does not matter whether the choice of which bit(s) to sample is performed by each client (local randomness) or prescribed by the server to each client (central randomness). Since the local setting is more vulnerable to clients who may try to poison the response by distorting the reported values of high-order bits, we favor central randomness.

3.2 Adaptive bit-pushing

A more sophisticated approach is to use a first round of bit-pushing to estimate the bit means $\bar{x}^{(j)}$. That is, we first choose a set of sampling probabilities p_j independent of the input, and ask a δ fraction of the clients to report an input bit according to this distribution. From these reports, we estimate $\bar{x}^{(j)}$ as $\hat{x}^{(j)}$ for all j, and use these estimates to compute a new set of weights based on $\beta'_j = \hat{x}^{(j)}(1 - \hat{x}^{(j)})4^j$. We can then perform a second round of bit-pushing using sampling probabilities $p_j = \sqrt{\beta'_j}/\sum_{j=0}^{b-1}\sqrt{\beta'_j}$ for the remaining $1 - \delta$ fraction of clients. To instantiate this two-round approach, we need to determine (*i*) what split parameter δ to apply; and (*ii*) how to choose the initial weights β_j. Naively, we might choose $\delta = \frac{1}{2}$ to balance accuracy of learned β'_js and accuracy of reported results. For β_js, we might default to choosing $\beta_j = 4^j$ (and hence $p_j \propto 2^j$), according to the above argument. Our full analysis [8] guides the choice of δ as $\frac{1}{3}$, and we will try different settings for both these choices in our empirical evaluations. Algorithm 2 provides pseudocode for the adaptive bit-pushing approach, where $*$ is used to denote scalar-vector multiplication. We next analyze the variance of two-round adaptive bit-pushing, and relate it to σ^2, the variance of the input.

Algorithm 2: Adaptive bit-pushing algorithm

```
Input  : No. of bits $b$, no. of clients $n$, parameters $\alpha$, $\gamma$, $\delta$
Output : Result $r$
/* Round 1:                                              */
for $j = 0$ to $b - 1$ do
    Compute $p_1[j] = (2^j)^\gamma$
Normalize $p_1$: $p_1 \leftarrow p_1/\text{sum}(p_1)$
Run basic bit-pushing:
    $(r_1, m_1, s_1) = \texttt{BitPushing}(b, p_1, \delta n)$
/* Round 2:                                              */
for $j = 0$ to $b - 1$ do
    Compute $p_2[j] = (4^j \cdot m_1[j] \cdot (1 - m_1[j]))^\alpha$
Normalize $p_2$: $p_2 \leftarrow p_2/\text{sum}(p_2)$
Run basic bit-pushing:
    $(r_2, m_2, s_2) = \texttt{BitPushing}(b, p_2, (1 - \delta)n)$
/* Final aggregation:                                    */
Combine means $m_3 = (s_1 + s_2)/(\delta n * p_1 + (1 - \delta)n * p_2)$
for $j = 0$ to $b - 1$ do
    $r \leftarrow r + 2^j \cdot m_3[j]$
return $r$
```

Lemma 3.4. *The variance of adaptive bit-pushing is bounded by* $O\left(C_{b_{\max}}\left(\frac{\sigma^2}{n} + \frac{4^{b_{\max}}}{n^{3/2}}\right)\right)$*, where* $b_{\max}$ *is the index of the highest-order bit that is non-zero in the input and* $C_{b_{\max}}$ *is a scaling factor.*

Assuming that each input bit has constant variance (discussed in the proof), this expression simplifies to $O(\sigma^2/n)$.

Comparison to alternate approaches. The benefit of adaptive bit-pushing can be most easily understood when we have only a loose estimate of b, the number of bits to represent the input values. For methods which scale the input down to the range $[0, 1]$ and then scale the estimated fraction back up, the variance of the resulting estimate is proportional to $(2^b)^2/n$. For (non-adaptive) bit-pushing, it is proportional to $2^b\bar{x}/n$, as shown in (7). Since adaptive bit-pushing allows us to identify any bits j with $\bar{x}_j = 0$, we can bound the variance of the estimate by $2^{b_{\max}}\bar{x}/n$,[4] or use the above analysis to argue that the variance is proportional to $b_{\max}\sigma^2/n$ plus lower order terms. Compared to the bound (7) for our non-adaptive protocol, variance is reduced by a factor proportional to $2^{b-b_{\max}}$.

Caching. Rather than producing our final estimate based solely on the reported values from the second round, we can pool the reports from both rounds. The net effect will be to gain more reports for each bit index, which should only improve the observed accuracy. We evaluate this "caching" technique in practice in Section 4.

3.3 Formal privacy guarantees

Reducing the number of bits transmitted about private data embodies the concept of data minimization, and provides an intuitive

[4] Note that $2^{b_{\max}}$ is a perhaps unintuitive quantity in this expression, and arises from pessimistically assuming that sampling weights are proportional to $2^j/(2^{b_{\max}})$.

notion of privacy to users. This can be augmented with more formal notions of privacy, to provide a strict guarantee.

Secure aggregation. Using a secure aggregation primitive ensures that the server knows the sum of the input values, without revealing anything further about the inputs of individual clients [26]. This can naturally apply to the bit-pushing approach, where the server can gather the sum and count of each bit location, and derive the mean from this information. However, this does not yet provide a differential privacy guarantee, for which extra steps are required.

Local differential privacy. Given a (private) bit y, *randomized response* is a simple procedure to mask the bit: with (public) probability p, we report y, otherwise we report $\bar{y} = 1 - y$. To unbias this report, we replace a reported value r with $\frac{r-(1-p)}{2p-1}$. If we run this procedure with $p = \frac{\exp\epsilon}{1+\exp\epsilon}$, the mechanism achieves the ϵ (local) differential privacy guarantee [14]. It is straightforward to give bit-pushing an ϵ-LDP guarantee: we apply randomized response to each bit before it is sent, and unbias the results at the server side. The variance of this unbiased estimator is $\frac{\exp\epsilon}{(\exp\epsilon-1)^2}$ [29]. In contrast to the above analysis, this variance is independent of the bit means $\bar{x}^{(j)}$. When we apply randomized response to bit-pushing, we assign $p_j n$ clients to report on bit j, which is scaled by 2^j. Hence, we bound the variance by $\sum_{j=0}^{b-1} \frac{4^j}{p_j n} \frac{\exp\epsilon}{(\exp\epsilon-1)^2}$. This is optimized following the above argument by setting $p_j = 2^j/(2^b - 1)$, producing a total variance bound of $O(\frac{4^b}{n} \frac{\exp\epsilon}{(\exp\epsilon-1)^2})$. Note that for small $\epsilon < 1$, this bound is $O(\frac{4^b}{n} \cdot \frac{1}{\epsilon^2})$, and the expected absolute error is $O(2^b/\epsilon\sqrt{n})$.

Because of the DP noise, we cannot rely on the bit means of unused bits to be zero. Instead, we apply filtering to determine which bits are mostly noise and should have their weight reduced. We can apply a simple heuristic of "bit squashing": if the value of a bit mean is below an absolute threshold, we assume that this bit is capturing noise and 'squash' it (i.e., downweigh its importance).

Distributed privacy guarantees. Local differential privacy can entail a large amount of noise, since each client adds sufficient noise to mask their own contribution. In the distributed privacy case, we instead have each client add only a small amount of noise, so that the aggregation of the noise is comparable to that added in the central differential privacy model. The data gathered in bit-pushing protocols is essentially a collection of binary histograms (counts of 0 and 1 bits for each bit index), for which accurate protocols exist under distributed privacy. Randomized response already provides a stronger distributed privacy guarantee [6], while Balcer and Cheu describe a similar approach based on Bernoulli noise addition [2]. More recently, Bharadwaj and Cormode show that random sampling is sufficient to give differential privacy, provided that very small counts are removed from the reporting [5]. The expected absolute error for a histogram due to the Bernoulli noise addition is $O(\frac{1}{\epsilon^2 n} \log 1/\delta)$ to give (ϵ, δ)-differential privacy when there are n clients. After scaling to estimate the mean, we obtain an error bound of $O(\frac{2^b}{\epsilon^2 n} \log(1/\delta))$, with an improved dependence on n compared to the local case.

3.4 Variance estimation

The empirical variance of a collection of data items is another fundamental aggregate. For instance, having estimates of the mean and the variance immediately enables *feature normalization* in federated learning. Computing the variance of the inputs, σ^2, can be reduced to two mean estimations of derived values: $\mathbb{V}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. These two expressions are equivalent when evaluated exactly but behave differently when used for estimation, where sample means replace expectations.

Lemma 3.5. *The estimated variance with bit-pushing via sample means in* $\mathbb{E}[(X - \mathbb{E}[X])^2]$ *has variance proportional to* $(\sigma^2 + \bar{x}^2/n)^2/n$, *while the estimated variance with bit-pushing via sample means in* $\mathbb{E}[X^2] - (\mathbb{E}[X])^2$ *has variance proportional to* $(\sigma^2 + \bar{x}^2)^2/n$.

Other functions, e.g., higher moments, products and geometric means, can also be approximated via bit-pushing; see [8] for details.

4 VALIDATION AND DEPLOYMENT

We now present our validation of the algorithm in practice and its deployment in industry infrastructure. Offline tests rely on Python implementations to simulate the computational costs and accuracy of mean estimation, and study performance scaling. Along with bit-pushing, we also implement methods based on piecewise-constant output distribution [28] ("piecewise"), and subtractive dithering [3] ("dithering"). Other DP methods (e.g., randomized rounding and Laplace noise) exhibited errors 2-3 times larger in all cases, so are omitted from the plots. We compare leading prior methods to two-round adaptive bit-pushing ("adaptive") and the single-round approach with fixed allocation of weights to bits ("weighted").

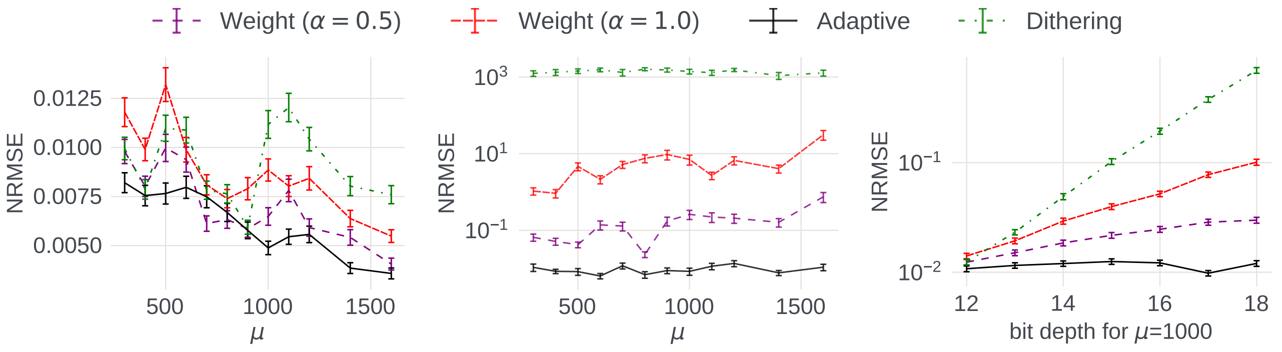
Our initial focus is on comparing the accuracy of the different techniques, measured by the normalized root-mean-squared error (NRMSE): in each experiment, we compare the true (empirical) value of the mean μ to the estimate $\hat{x}$, and compute the mean of the squared difference over 100 independent repetitions, then divide by the true mean μ for normalization. We perform experiments on both human-generated and synthetic data. To study performance scaling, we generated synthetic data by drawing values from Normal, uniform and exponential distributions with varying parameters, as specified below. The human-generated data (*census data* for short) are the distribution of people's ages from publicly-available US Census data.[5] We only compute the mean age and the variance of ages to check how well our methods work. Error bars on our plots indicate the standard error. Our default number of clients — 10K — is representative of typical federated analytics scenarios. The offline experiments in Sections 4.1 and 4.2 are easy to reproduce, reconfigure, and scale. In Section 4.3, they are complemented by qualitative findings from real-time product data.

4.1 Accuracy experiments

Figure 1 shows evaluation of accuracy for mean and variance. Here, we evaluate the *subtractive dithering* approach [3] as an alternative approach which reveals only one bit of information about the client's value. We compare against the single-round and adaptive bit-pushing techniques, with default parameters $\gamma = 0.5$, $\delta = 1/3$, and testing both $\alpha = 0.5$ and $\alpha = 1$.

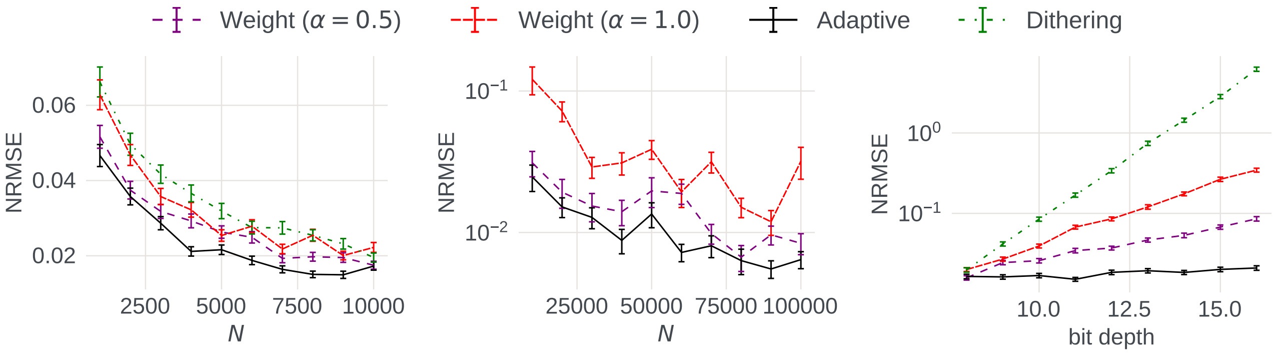
Figure 1a shows the accuracy as we vary the mean of the input (Normal) distribution. There is a general trend for the normalized error to decrease, as the normalizing constant increases faster

[5] https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29 — see [12].



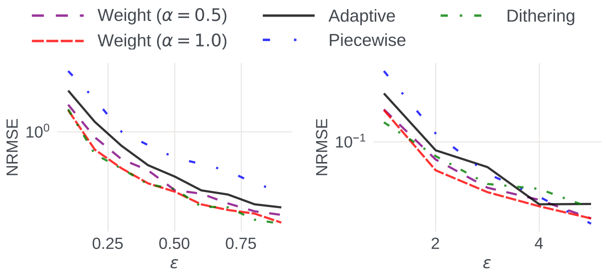
(a) Estimating mean with μ varying **(b) Estimating variance with μ varying** **(c) Estimating mean with varying bit depth**

Figure 1: Accuracy experiments on Normal distributed data with standard deviation $\sigma = 100$



(a) Estimating mean with varying n **(b) Estimating variance with varying n** **(c) Estimating mean with varying bit depth**

Figure 2: Accuracy experiments on census data



(a) Mean estimation, $\epsilon < 1$ **(b) Mean estimation, $\epsilon > 1$**

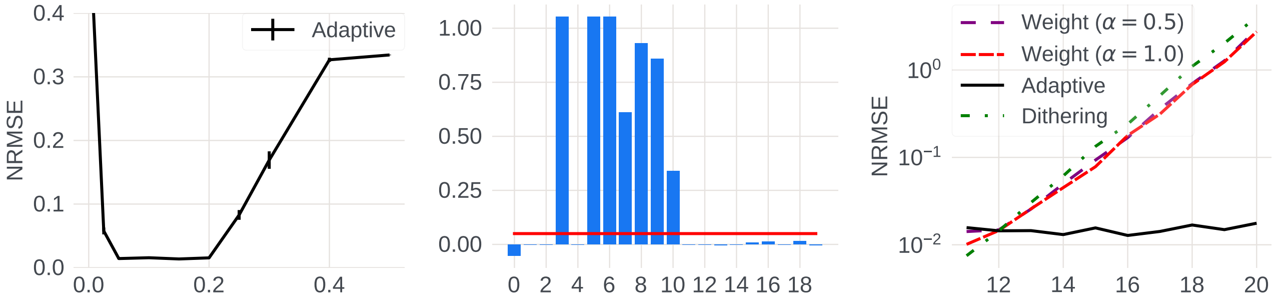
Figure 3: Differential privacy experiments on census data

than the magnitude of the errors. This is most notable for the dithering approach around powers of two, where there is a step up in error at the point where we increase the bound on the input values by a factor of 2. As in the previous experiments, in a single round, choosing weights based on $\alpha = 0.5$ generally leads to more accurate results. Across the whole domain, we find that the adaptive approach reliably achieves the least error.

We see a different set of behaviors when we attempt to estimate the variance of the distribution, in Figure 1b. This is a harder task, as evidenced by error values, which are substantially larger. We allocate a larger cohort of 100,000 clients to this task. Here, the dithering approach is orders of magnitude worse, due to its inability to adapt to the scale of the input values. Among the weighted approaches, it is now the $\alpha = 0.5$ results that are preferred. However, the adaptive approach achieves the best accuracy. The larger user cohort moderates the errors: the adaptive approach keeps the normalized errors down in the 1-2% range. Increasing the number of participating clients n would improve the accuracy further.

We further study the importance of finding accurate bounds on the magnitude of the quantities involved in Figure 1c. We vary the "bit depth", which is the number of bits b used in the bit-pushing algorithms (so 2^b is the bound used for the dithering approach). We see that all the one-round approaches grow in error as b increases: less so for $\alpha = 0.5$, since less weight is apportioned to the (vacuous) high order bits than in the $\alpha = 1.0$ case. The adaptive approach, meanwhile, is able to identify the redundant bits in the first round, and discards them in round two. Hence, it is largely oblivious to the increase in bit depth.

A corresponding set of experiments varying n on census data are shown in Figure 2. As expected, the normalized error for both mean (Figure 2a) and variance (Figure 2b) estimation tends to decrease as n increases, broadly consistent with the predicted dependence on $n^{-1/2}$. Unsurprisingly, there is some fluctuation, with the adaptive approach showing more variability for smaller values of n for variance estimation. Again, we see that the adaptive approach handles the increasing number of bits the best of the methods (Figure 2c).



(a) Varying the threshold for *bit squashing*

(b) Histogram of bit means

(c) Varying bit depth under DP, $\epsilon = 2.0$

Figure 4: Accuracy experiments on synthetic data with differential privacy

4.2 Differential privacy tradeoffs

In this section, we study the impact of providing a differential privacy (DP) guarantee on mean estimation. We additionally consider the "piecewise" mechanism [28], as well as our previous one-bit methods augmented with randomized response to provide a DP guarantee in the local model. Figure 3 shows the RMSE accuracy as we vary the privacy parameter ϵ, split into two regimes: high privacy ($\epsilon < 1$), Figure 3a, and moderate privacy ($\epsilon \geq 1$, Figure 3b). We omit results for Laplace noise, where the observed error was considerably higher than others, as expected. On a log-scale plot, the lines are fairly closely clustered, but we see that in this experiment, the single round approach with $\alpha = 1.0$ achieves the least error. Only when $\epsilon > 3$ do we see points where the adaptive and piecewise approaches achieve lower error. Note that the absolute value of RMSE is an order of magnitude larger than without DP noise. This is consistent with our theoretical analysis, where we showed that the variance depends on ϵ (as $\frac{\exp \epsilon}{(\exp \epsilon - 1)^2}$), and is independent of the value of the bit means. Hence, the adaptive approach (focusing on bits with higher variance) holds no advantage here.

Figure 4a shows the effect on RMSE as we vary the threshold for bit squashing (Section 3.3), as a multiple of the expected amount of DP noise. It turns out that applying a threshold of 0.05–0.2 is very effective at improving accuracy by almost two orders of magnitude. Figure 4b shows an example in more detail, with a histogram of the estimated bit means for the noisy data with $\epsilon = 2$, along with the threshold of 0.05. We see that the DP noise causes some of these estimates to exceed 1.0 or fall below 0.0 (when the DP subtrahend exceeds the true mean). However, there is a clear "dense" region up to bit 10, with higher bits showing random noise. The bit-squashing approach treats bits 11 and above as noise, and bases the estimate on bits 0-10 only. Figure 4c shows this in practice as we increase the bit depth: the adaptive approach using bit squashing maintains the same level of accuracy, while all other methods grow in error proportional to the magnitude of the (noisy) values.

4.3 Deployment experience

Having quantified the performance and scaling of bit-pushing through our offline tests, we implemented it online in the Federated Analytics (FA) software stack at Meta, which is part of the Papaya platform for Federated Learning [17]. As an alternative to industry-standard aggregation, bit-pushing does not use additional data or compute additional types of statistics. However, it reduces private data sharing from entire values to a single (binary) digit per value. We implement differential privacy according to common industry practice: by (*i*) enforcing local differential privacy via client-side randomized data transforms, (*ii*) adding distributed noise via sampling [5] (see Section 3.3 for details). Our deployment of bit-pushing aggregates *device health and performance metrics* to monitor summary statistics.[6] Different metrics imply different value distributions and exercise different cornercases for our aggregation protocols, providing a comprehensive online evaluation, as well as comparisons to other aggregation techniques. We detail insights from the deployment evaluation of the bit-pushing algorithm over several months with emphasis on numerical accuracy, robustness and latency.

Deciding the number of bits. Our deployment experience underlined the importance of calibrating data aggregation protocols to the extremely heterogeneous data distributions found in the wild, which can be very different from analytically-modeled statistical distributions. For several metrics, we saw distributions with extreme outliers. For example, when observing features whose most typical values are 0 and 1, we might see a few single-digit values, but some rare clients report values that are orders of magnitude higher. Any of the methods considered for mean estimation will struggle with such extreme distributions and produce inaccurate estimates. Notably, estimating the mean might not be appropriate for such data: the sample mean is very sensitive to which outlier clients respond and share their data. *Robust statistics* are more appropriate, such as the median and percentiles, but a first approach is to apply *winsorization* to the input, such as via *clipping*. In conjunction with bit-pushing, this can be realized by clipping the inputs to a fixed number of bits b – say, 8 or 16 – so that large values are truncated to $2^b - 1$. Leveraging domain knowledge to choose the appropriate number of bits leads to good accuracy in practice, without having to know tight bounds on the values that might appear in practice. At the other extreme, some metrics/features gathered turn out to be constant, making mean and variance estimation moot. Such checks can be performed offline.

Performance of aggregation protocols on unexpected data distributions is particularly relevant to *federated debugging*, where distributed data can be affected by misconfigurations or mistakes

[6] As per Sections 1.1 and 3, at most one binary digit is shared for each sensitive value, and this bit is perturbed by randomized response [31] to offer plausible deniability, over and above the security guarantees of secure aggregation.

in software. In such cases, diagnosis of problematic issues is complicated by the inability to read distributed private data.

Impact of noise. After clipping, the accuracy of online mean estimation matched those observed during offline validation, and errors relative to ground-truth data collection were small. Hence, offline simulations are sufficient to set the parameters for online noise. Likewise, the inclusion of privacy noise in deployment increases uncertainty correspondingly to the offline case. ① We observed empirically that *local differential privacy* does increase the magnitude of variability in results, but not so much as to change the conclusions drawn from noise-free data analysis. ② When many high-order bits do not contain information of value, the adaptive approach reduces the observed error by significant factors. ③ We also found that achieving a *central differential privacy* guarantee by having the enclave apply thresholding to the reported bit counts was effective, and introduced a negligible amount of noise compared to the non-thresholded sample [5].

Robustness with respect to intermittent connectivity. Client devices participating in FA exhibit diverse system characteristics, and their network connection can be unreliable. Two properties of the bit-pushing algorithm make it attractive in practice: ① The algorithm succeeds even with a small subset (10s of thousands) of devices responding to queries, compared to the scale of the entire device population. It does not require all devices to be available at query time. ② It is not overly sensitive to the bit-sampling probability. Client devices can drop out at any point of the federated protocol, so to accommodate this, the bit sampling probabilities were auto-adjusted based on the dropout rate, improving utility.

Latency and number of rounds. Federated data collection provides strong guarantees of accuracy and privacy, but comes at the cost of increased latency to gather data. When the devices are available to communicate with the server, they are assigned a compute task, perform this computation on local data, and report back the result, sometimes executing multiple rounds of a protocol. The typical time to complete a round on our FA stack is a matter of minutes, so even adaptive bit-pushing which performs two rounds of data collection is fast, particularly over statistics that are readily available (e.g., represent the previous day's activity). However, when applied to more selective queries, e.g., restricting eligibility to clients in a particular geography, it can take longer for a sufficient number of eligible clients to make themselves available. Here, it is pertinent to achieve good accuracy as a function of number of participants, and to enforce a minimum cohort size for privacy.

Aggregating multiple local values per feature. The literature on federated analytics typically describes algorithms that take a single value from each client [1, 24, 32]. However, for many features of interest, most clients hold several values (e.g., device parameter readings at different times), while a small subset may hold up to millions of observations. Handling this circumstance is largely a matter of semantics: we could choose to elicit a single value from each client by sampling *or* local aggregation, or else we could provide a multiset or weighted response. A discrepancy is possible if we define *ground truth* using the mean of all client's values, but compute a federated mean based on sampling: clients with many values differ in value distribution from the rest of the population. In our setting, it is appropriate to aggregate a single value per client, so we define the ground truth for data collection via sampling (taking the mean produced similar results).

5 CONCLUSIONS AND PERSPECTIVES

Performing accurate, private analytics is vital in many applied industrial contexts. Measurement needs at Meta motivated our study of practical private numerical aggregation algorithms. Our mathematical results and our offline evaluation show that bit-pushing offers high numerical accuracy while sharing at most one bit of a client's value to the server using industry-standard randomized response [31]; bit-pushing greatly outperforms prior techniques when aggregated values are in a narrow range unknown in advance.

Notions of optimality for single-bit estimates have been explored in the recent literature, and methods such as *subtractive dithering* were shown optimal [3]. Hence, it may look surprising that bit-pushing empirically outperforms those methods. This apparent paradox stems from the assumptions made in prior proofs of optimality. In particular, optimality is invalidated when the true mean can be narrowed down further within a fixed subrange $[0, 1]$, and this bracketing is performed by adaptive bit-pushing using the same type of inputs used by other protocols. In other words, prior optimality results can provide loose bounds in practice, and in some practical cases we improve accuracy by orders of magnitude.

Limitations of the approach are the flip side of its advantages: when a tight bound on the values *is* known in advance, then bit-pushing and prior methods attain similar accuracy. However, accuracy is not the only relevant metric. When a single bit of the client's input does not reveal sensitive information, bit-pushing alone can provide an intuitive privacy promise to non-experts. When some bits of a value can be privacy-revealing (say, disclosing whether a value is above or below a threshold), plausible deniability for communicated bits is ensured using differential privacy via noise [31].

Communication costs of our approach are low, since only a single private bit of data is disclosed. However, there are additional overheads to include header information, and list which bit was sampled, so the distinction between sending a single bit versus a few numeric values is not so meaningful: both can be easily communicated within a single (encrypted) network packet. In settings where each client sends multiple bits, or reveals information about multiple features, the communication benefits become more apparent.

Robustness to poisoning attacks is a concern for many LDP algorithms [2, 7]. Since the result averages over all client reports, no one client can influence the outcome significantly. However, if clients choose which bit to report, then an adversarial client could pick the most significant bit ($b_{\max}$) and deterministically send a 1 (say), to bias the result upwards. In this setting, central randomness, where the server picks the bits to report (Section 3.1), reduces the impact of such poisoning attacks.

Trustworthy privacy infrastructure is needed to support novel federated analytics protocols. We have taken one such algorithm from theory to practice in the Papaya platform [17]. Our approach limits private bits shared by the device, but the device will naturally share additional (non-private) bits to implement the protocol. Platforms for federated analytics and learning could provide configurable aggregation services that would package private bits into larger network packets and provide layers of separation to rule out the mixing of private and non-public bits. With such infrastructure, it can be relatively easy to add privacy metering and monitoring, potentially giving the users greater control of their data privacy.

Acknowledgments. We are grateful to colleagues who provided input and feedback on this work, including Ilya Mironov, Akash Bharadwaj, Kaikai Wang, Peng Chen, and Dzmitry Huba.

REFERENCES

[1] Eugene Bagdasaryan et al. 2021. Towards Sparse Federated Analytics: Location Heatmaps under Distributed Differential Privacy with Secure Aggregation. *CoRR* abs/2111.02356 (2021). arXiv:2111.02356 https://arxiv.org/abs/2111.02356

[2] Victor Balcer, Albert Cheu, Matthew Joseph, and Jieming Mao. 2021. Connecting Robust Shuffle Privacy and Pan-Privacy. In *Proc. Symp. Discrete Algorithms, SODA*. SIAM. https://doi.org/10.1137/1.9781611976465.142

[3] Ran Ben-Basat, Michael Mitzenmacher, and Shay Vargaftik. 2020. How to send a real number using a single bit (and some shared randomness). arxiv:2010.02331. arXiv:2010.02331 https://arxiv.org/abs/2010.02331

[4] Akash Bharadwaj and Graham Cormode. 2022. An Introduction to Federated Computation. In *SIGMOD International Conference on Management of Data*. ACM, 2448–2451. https://doi.org/10.1145/3514221.3522561

[5] Akash Bharadwaj and Graham Cormode. 2022. Sample and Threshold Differential Privacy: Histograms and applications. In *AISTATS*.

[6] Albert Cheu. 2021. Differential Privacy in the Shuffle Model: A Survey of Separations. *CoRR* abs/2107.11839 (2021). arXiv:2107.11839 https://arxiv.org/abs/2107.11839

[7] Albert Cheu, Adam D. Smith, and Jonathan R. Ullman. 2021. Manipulation Attacks in Local Differential Privacy. *J. Priv. Confidentiality* 11, 1 (2021). https://doi.org/10.29012/jpc.754

[8] Graham Cormode and Igor L. Markov. 2021. Bit-efficient Numerical Aggregation and Stronger Privacy for Trust in Federated Analytics. *CoRR* abs/2108.01521 (2021). arXiv:2108.01521 https://arxiv.org/abs/2108.01521

[9] Differential Privacy Team at Apple. [n.d.]. Learning with Privacy at Scale. https://machinelearning.apple.com/research/learning-with-privacy-at-scale.

[10] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *NeurIPS*. 3571–3580. https://proceedings.neurips.cc/paper/2017/hash/253614bbac999b38b5b60cae531c4969-Abstract.html

[11] Josep Domingo-Ferrer, David Sánchez, and Alberto Blanco-Justicia. 2021. The limits of differential privacy (and its misuse in data release and machine learning). *Comm. ACM* 64, 7 (2021), 33–35. https://doi.org/10.1145/3433638

[12] D. Dua and C. Graff. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. University of California Irvine.

[13] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. 2018. Minimax optimal procedures for locally private estimation. *J. Amer. Statist. Assoc.* 113, 521 (2018), 182–201.

[14] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014).

[15] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proc. ACM SIGSAC Conf. Comput. and Commun. Security*. ACM, 1054–1067. https://doi.org/10.1145/2660267.2660348

[16] Andy Greenberg. 2017. How One of Apple's Key Privacy Safeguards Falls Short. https://www.wired.com/story/apple-differential-privacy-shortcomings. *Wired* (September 2017).

[17] Dzmitry Huba et al. 2022. PAPAYA: Practical, Private, and Scalable Federated Learning. In *MLSys*. mlsys.org. https://arxiv.org/abs/2111.04877

[18] Intel Specialized Development Tools. [n.d.]. Software Guard Extensions. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[19] Dan Levy. 2020. Speaking Up for Small Businesses. https://www.facebook.com/business/news/ios-14-apple-privacy-update-impacts-small-business-ads. Facebook for Business.

[20] Zhi-Quan Luo. 2005. Universal decentralized estimation in a bandwidth constrained sensor network. *IEEE Trans. Inf. Theory* 51, 6 (2005), 2210–2219. https://doi.org/10.1109/TIT.2005.847692

[21] Brendan McMahan and Daniel Ramage. 2017. Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html. Google AI Blog.

[22] Paul Ohm. 2009. Broken Promises of Privacy: Responding to the Surprising Failure of Anonymization. *UCLA Law Review, Vol. 57, p. 1701, 2010* (2009).

[23] Matthias Paulik and et al. 2021. Federated Evaluation and Tuning for On-Device Personalization: System Design & Applications. `arXiv:2102.08503`.

[24] Daniel Ramage and Stefano Mazzocchi. 2020. Federated Analytics: Collaborative Data Science without Data Collection. https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html. Google AI Blog.

[25] Mike Schneider. 2021. Groups: Census privacy tool could hurt voting rights goals. https://apnews.com/general-news-907d94c8e280b173dc2942feda181348. Associated Press.

[26] Aaron Segal et al. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*. https://eprint.iacr.org/2017/281.pdf

[27] Royal Society. 2021. Privacy Enhancing Technologies report. https://royalsociety.org/-/media/policy/projects/privacy-enhancing-technologies/privacy-enhancing-technologies-report.pdf.

[28] Ning Wang, Xiaokui Xiao, Yin Yang, Jun Zhao, Siu Cheung Hui, Hyejin Shin, Junbum Shin, and Ge Yu. 2019. Collecting and Analyzing Multidimensional Data with Local Differential Privacy. In *IEEE Int'l Conf. Data Eng.* IEEE, 638–649. https://doi.org/10.1109/ICDE.2019.00063

[29] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally Differentially Private Protocols for Frequency Estimation. In *USENIX Security Symp.* USENIX Association, 729–745. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tianhao

[30] Teng Wang, Xuefeng Zhang, Jinyu Feng, and Xinyu Yang. [n.d.]. A Comprehensive Survey on Local Differential Privacy: Toward Data Statistics and Analysis. `arXiv:2010.05253`.

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

[32] Wennan Zhu et al. 2020. Federated Heavy Hitters Discovery with Differential Privacy. In *AISTATS*, Vol. 108. PMLR, 3837–3847. http://proceedings.mlr.press/v108/zhu20a.html