Machine Learning Journal

Learning with Privacy at Scale

Vol. 1, Issue 8 • December 2017

by Differential Privacy Team

Understanding how people use their devices often helps in improving the user experience. However, accessing the data that provides such insights — for example, what users type on their keyboards and the websites they visit — can compromise user privacy. We develop a system architecture that enables learning at scale by leveraging local differential privacy, combined with existing privacy best practices. We design efficient and scalable local differentially private algorithms and provide rigorous analyses to demonstrate the tradeoffs among utility, privacy, server computation, and device bandwidth. Understanding the balance among these factors leads us to a successful practical deployment using local differential privacy. This deployment scales to hundreds of millions of users across a variety of use cases, such as identifying popular emojis, popular health data types, and media playback preferences in Safari. We provide additional details about our system in the full version.

Introduction

Gaining insight into the overall user population is crucial to improving the user experience. The data needed to derive such insights is personal and sensitive, and must be kept private. In addition to privacy concerns, practical deployments of learning systems using this data must also consider resource overhead, computation costs, and communication costs. In this article, we give an overview of a system architecture that combines differential privacy and privacy best practices to learn from a user population.

Differential privacy [2] provides a mathematically rigorous definition of privacy and is one of the strongest guarantees of privacy available. It is rooted in the idea that carefully calibrated noise can mask a user's data. When many people submit data, the noise that has been added averages out and meaningful information emerges.

Within the differential privacy framework, there are two settings: *central* and *local*. In our system, we choose not to collect raw data on the server which is required for central differential privacy; hence, we adopt local differential privacy, which is a superior form of privacy [3]. Local differential privacy has the advantage that the data is randomized before being sent from the device, so the server never sees or receives raw data.

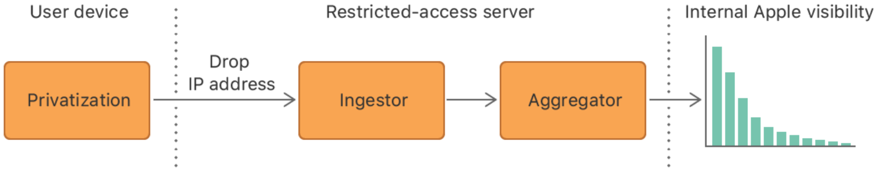
Our system is designed to be opt-in and transparent. No data is recorded or transmitted before the user explicitly chooses to report usage information. Data is privatized on the user's device using event-level differential privacy [4] in the local model where an event might be, for example, a user typing an emoji. Additionally, we restrict the number of transmitted privatized events per use case. The transmission to the server occurs over an encrypted channel once per day, with no device identifiers. The records arrive on a restricted-access server where IP identifiers are immediately discarded, and any association between multiple records is also discarded. At this point, we cannot distinguish, for example, if an emoji record and a Safari web domain record came from the same user. The records are processed to compute statistics. These aggregate statistics are then shared internally with the relevant teams at Apple.

We focus on the problem of estimating frequencies of elements — for example, emojis and web domains. In estimating frequencies of elements, we consider two subproblems. In the first, we compute the histogram from a *known* dictionary of elements. In the second, the dictionary is *unknown* and we want to obtain a list of the most frequent elements in a dataset.

System Architecture

Our system architecture consists of device-side and server-side data processing. On the device, the *privatization* stage ensures raw data is made differentially private. The restricted-access server does data processing that can be further divided into the *ingestion* and *aggregation* stages. We explain each stage in detail below.

Figure 1. System Overview.

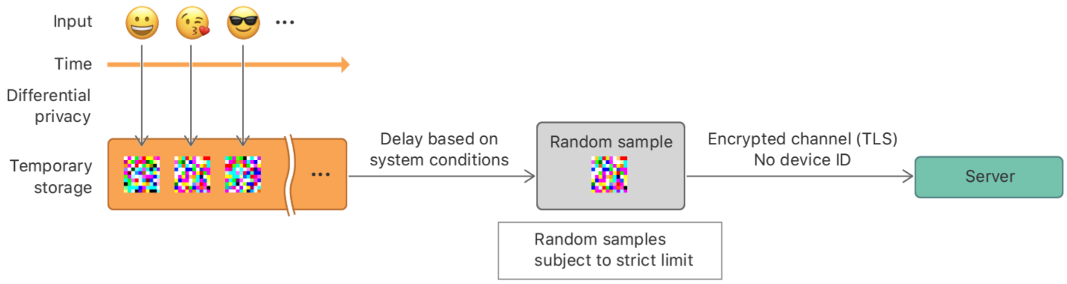


Privatization

Users have the option, in System Preferences on macOS or in Settings on iOS, to share privatized records for analytics. For users who do not opt in, the system remains inactive. For users who do opt in, we define a per-event privacy parameter, ϵ. Additionally, we set a limit on the number of privatized records that can be transmitted daily for each use case. Our choice of ϵ is based on the privacy characteristics of the underlying dataset for each use case. These values are consistent with the parameters proposed in the differential privacy research community, such as [5] and [6]. Moreover, the algorithms we present below provide users further deniability due to hash collisions. We provide additional privacy by removing user identifiers and IP addresses at the server where the records are separated by use case so that there is no association between multiple records.

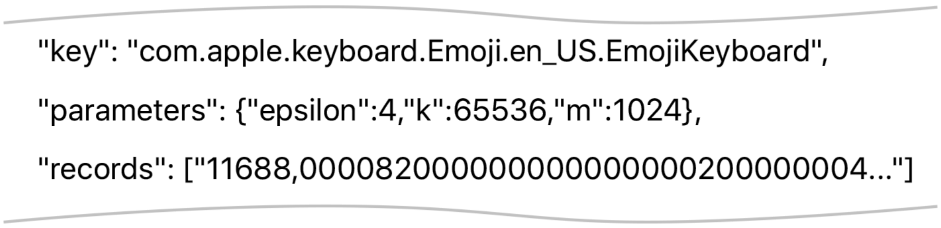
Whenever an event is generated on device, the data is immediately privatized via ϵ-local differential privacy and temporarily stored on-device using data protection [1], rather than being immediately transmitted to the server. After a delay based on device conditions, the system randomly samples from the differentially private records subject to the above limit and sends the sampled records to the server. These records do not include device identifiers or timestamps of when events were generated. The communication between a device and the server is encrypted using TLS. See Figure 2 for an overview.

Figure 2. Privatization Stage.



On iOS, the reports are visible in Settings > Privacy > Analytics > Analytics Data in entries that begin with "DifferentialPrivacy." On macOS, these records are visible in Console, in System Reports. Figure 3 is a sample record of our algorithm for the Popular Emojis use case. The record lists algorithmic parameters, which are discussed in the section below, and the privatized data entry is represented as a hexadecimal string. Note that the privatized data is elided here for presentation; the full size in this example is 128 bytes.

Figure 3. Sample Report with Privatized Record.



```
"key": "com.apple.keyboard.Emoji.en_US.EmojiKeyboard",
"parameters": {"epsilon":4,"k":65536,"m":1024},
"records": ["11688,0000820000000000000000200000004..."]
```

Ingestion and Aggregation

The privatized records are first stripped of their IP addresses prior to entering the ingestor. The ingestor then collects the data from all users and processes them in a batch. The batch process removes metadata, such as the timestamps of privatized records received, and separates these records based on their use case. The ingestor also randomly permutes the ordering of privatized records within each use case before forwarding the output to the next stage.

The aggregator takes the private records from the ingestor and for each use case generates a differentially private histogram according to the algorithms described in the section below. The data from multiple use cases is never combined when computing statistics. In these histograms, only domain elements whose counts are above a prescribed threshold T are included. These histograms are then shared internally with relevant teams at Apple.

Algorithms

We now describe three local differentially private algorithms in the following sections.

Private Count Mean Sketch

The Private Count Mean Sketch algorithm (CMS) aggregates records submitted by devices and outputs a histogram of counts over a dictionary of domain elements, while preserving local differential privacy. This happens in two stages: client-side processing followed by server-side aggregation.

We illustrate the process with an example. Suppose a user visits the web domain $www.example.com$. The client-side algorithm randomly samples a hash function from a set of candidate hash functions $\{h_1, h_2, h_3, \ldots, h_k\}$ and encodes the web domain into a small space of size m using the selected hash function, say h_2. Let $h_2(www.example.com) = 31$. This encoding is written as a one-hot vector of size m where the bit at the 31st position is set to 1. To ensure differential privacy, each bit of the one-hot vector is independently flipped with probability $\frac{1}{e^{\epsilon/2}+1}$, where ϵ is the privacy parameter, which forms the privatized vector. This vector and the selected hash function index is then sent to the server.

The server-side algorithm constructs a *sketch matrix* M by aggregating the privatized vectors from the devices. The matrix has k rows — one for each hash function — and m columns corresponding to the size of the vector transmitted from the client.

As records arrive on the server, the algorithm adds the privatized vector to the vector at row j, where j is the index of the hash function sampled by the device. The values of M are then scaled appropriately so that each row helps provide an unbiased estimator for the frequency of each element.

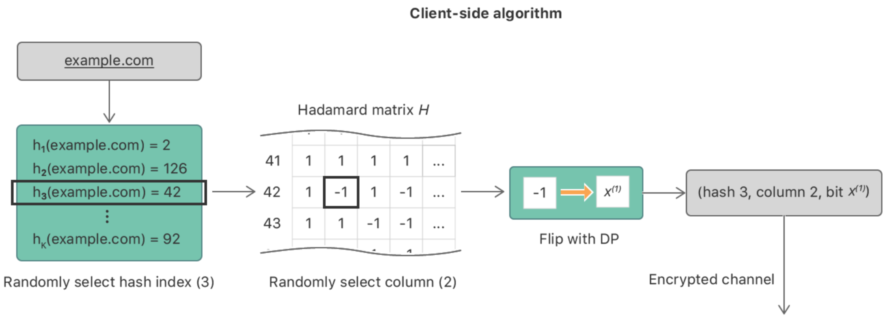
To compute the frequency for a web domain $www.example.com$, the algorithm takes each unbiased estimate by reading $M[j, h_j(www.example.com)]$ for each row j and computes the mean of these estimates. In the full version of this article, we prove an analytic expression for the error (or variance) of the private counts, which allows us to have a principled approach in obtaining accurate counts while minimizing resource overhead like device bandwidth and server runtime in our deployment.

Private Hadamard Count Mean Sketch

We describe in the full version of this article how increasing the device bandwidth will lead to more accurate counts in CMS. However, this leads to higher transmissions costs to users. We wanted to have a minimal impact on accuracy while reducing transmission cost. This led us to design the Private Hadamard Count Mean Sketch algorithm (HCMS), which has the advantage that the device can send one bit with a small loss in accuracy. With HCMS, it is possible to achieve reasonably accurate counts without having users pay a high transmission cost. We quantify the accuracy that we obtain with HCMS in the full version.

We now present HCMS with an example. Suppose a user visits the web domain $www.example.com$. As in CMS, the client-side algorithm selects a random hash function from a set of candidate hash functions $\{h_1, h_2, h_3, \ldots, h_k\}$ and encodes the web domain into a small space using the selected hash function, say h_3. Let $h_3(www.example.com) = 42$. This encoding is written as a one-hot vector $\mathbf{v} = (0, 0, \ldots, 0, 1, 0, \ldots, 0, 0)$ where the 1 is at position 42. Since we want to transmit a single bit, a trivial approach would be to sample and send a random coordinate from $\mathbf{v}$. However, this increases the error (variance) in the resulting histogram significantly. To help reduce the variance, we use the Hadamard basis transform H on $\mathbf{v}$ to obtain $\mathbf{v}' = H\mathbf{v} = (+1, -1, \ldots, +1)$, for instance. A single random coordinate is sampled from $\mathbf{v}'$, and the corresponding bit is flipped with probability $\frac{1}{e^{\epsilon}+1}$, to ensure differential privacy. The output sent to the server includes the index of the selected hash function, the sampled coordinate index and the privatized bit; see Figure 4.

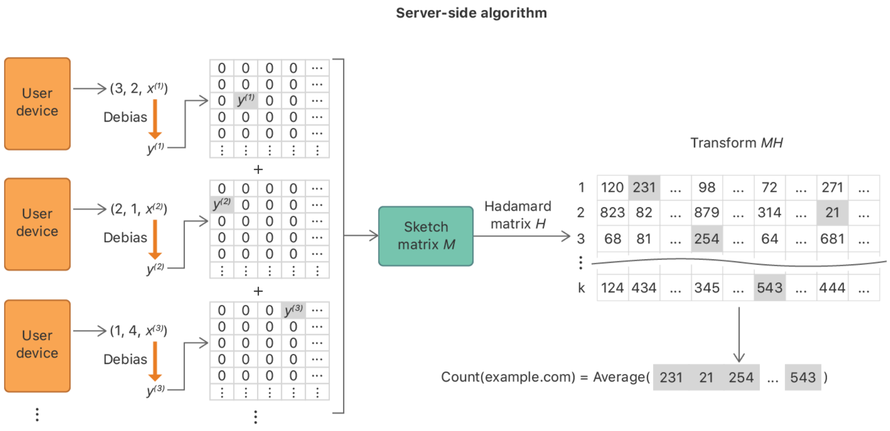
Figure 4. Client-Side Algorithm for Hadamard Count Mean Sketch.



41	1	1	1	1	...
42	1	-1	1	-1	...
43	1	1	-1	-1	...

Similar to CMS, the server-side algorithm uses a data structure, *sketch matrix* M to aggregate the privatized vectors from the clients. The rows of the matrix M are indexed by the candidate hash functions. Additionally, the columns are indexed by the random coordinate indices that the device samples. The (j, l)th cell of the matrix aggregates the privatized vectors submitted by devices that chose the jth hash function h_j and sampled the l th coordinate from the vector. Further, privatized vectors are scaled appropriately and M is transformed back to the original basis using the inverse Hadamard matrix. At this stage, each row of the matrix helps provide an unbiased estimator for the frequency of an element. To compute the frequency for a web domain $www.example.com$, the algorithm first obtains an estimate from each row of M by reading $M[j, h_j(www.example.com)]$ from row j. As a final step, the algorithm computes the mean of these k estimates to reduce variance; see Figure 5.

Figure 5. Server-Side Algorithm for Hadamard Count Mean Sketch.



Private Sequence Fragment Puzzle

The previous algorithms assume there is some known dictionary of domain elements through which the server can enumerate in order to determine the corresponding counts. However, in certain use cases, the domain is massive and enumerating over the full space is computationally prohibitive. For example, in discovering frequently typed new words, even if we restrict ourselves to 10-letter case-sensitive English words, this approach would require the server to loop over at least 52^{10} elements.

Instead, we develop an algorithm called Sequence Fragment Puzzle (SFP) and present it in the setting of discovering new words. We take advantage of the fact that given a popular string, any substring of that string is also at least as popular. On the device, we use the client-side CMS algorithm to privatize the typed word. Additionally, we select a substring of the word and concatenate it with an 8-bit hash of the word. We refer to the small hash as the *puzzle piece* and the substring concatenated with the hash as the *fragment*. The fragment is privatized using CMS and also transmitted to the server along with the privatized word. For example, if the word is $Despacito$ and the selected substring is sp, the client sends three things: CMS $(Despacito)$, CMS$(sp||97)$ where 97 is the puzzle piece, and the location of the selected substring.

Using sketches for fragments, the server-side algorithm obtains a histogram over all possible fragments for each substring location. The puzzle piece allows the server to correlate fragments from the same word since all fragments from a word will have the same puzzle piece. Then, restricting itself to the most popular fragments, the server algorithm determines a list of candidate strings by concatenating popular fragments whose puzzle pieces match. The set of candidate strings forms a dictionary of reasonable size and lets us use the CMS algorithm on the full word.

Results

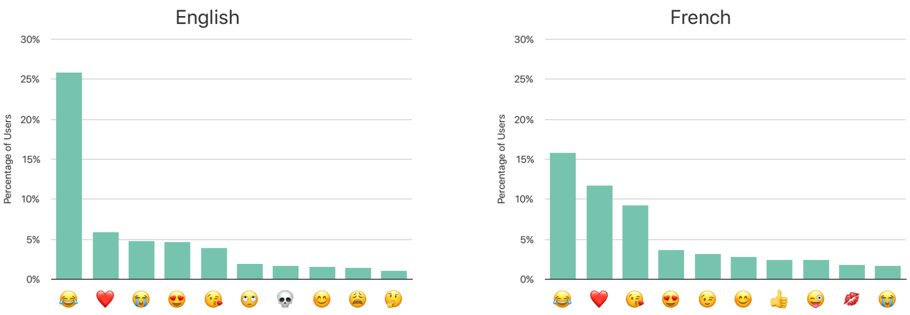
We present three use cases below to illustrate how our algorithms are used to enhance product features while protecting user privacy.

Discovering Popular Emojis

Given the popularity of emojis across our user base, we want to determine which specific emojis are most used by our customers and the relative distribution of these characters. To that end, we deploy our algorithms to understand the distribution of emojis used across keyboard locales. For this use case, we set the parameters for CMS to be m = 1024, k = 65,536, and ϵ = 4 with dictionary size of 2600 emojis.

The data shows many differences across keyboard locales. In Figure 6, we observe snapshots from two locales: English and French. Using this data, we can improve our predictive emoji QuickType across locales.

Figure 6. Emojis in Different Keyboard Locales.



Identifying High Energy and Memory Usage in Safari

Some websites are exceedingly resource-intensive, and we wish to identify these sites in order to ensure a better user experience. We consider two types of domains: those that cause high memory usage and those that cause excessive energy drain from CPU usage. In iOS 11 and macOS High Sierra, Safari can automatically detect these exceptional domains and report them using differential privacy.

Using our algorithms, we are able to determine which domains have high resource consumption. For this use case, we set the parameters for HCMS to be m = 32,768, k = 1024, and ϵ = 4 with dictionary size of 250,000 web domains. Recall that the differentially private record is only a single bit in HCMS. Our data shows that the most common, high-resource domains include video consumption websites, shopping websites, and news websites.

Discovering New Words

We want to learn words that are not present in the lexicons included on the device in order to improve auto-correction. To discover new words, we deploy the Sequence Fragment Puzzle (SFP) algorithm described above.

The algorithm produces results spanning several languages, including English, French, and Spanish. The learned words for the English keyboard, for example, can be divided into multiple categories: abbreviations like *wyd*, *wbu*, *idc*; popular expressions like *bruh*, *hun*, *bae*, and *tryna*, seasonal or trending words like *Mayweather*, *McGregor*, *Despacito*, *Moana*, and *Leia*; and foreign words like *dia*, *queso*, *aqui*, and *jai*. Using the data, we are constantly updating our on-device lexicons to improve the keyboard experience.

Another category of words discovered are known words without the trailing *e* (*lov* or *th*) or *w* (*kno*). If users accidentally press the left-most prediction cell above the keyboard, which contains the literal string typed thus far, a space will be added to their current word instead of the character they intended to type. This is a key insight that we were able to learn due to our local differentially private algorithm.

Conclusion

In this article, we have presented a novel learning system architecture, which leverages local differential privacy and combines it with privacy best practices. To scale our system to millions of users and a variety of use cases, we have developed novel local differentially private algorithms – CMS, HCMS, and SFP – for both the known and unknown dictionary settings. In our full paper, we have provided analytic expressions for the tradeoffs among various factors, including privacy, utility, server computation overhead, and device bandwidth. Our utility theorems give a principled way to choose algorithmic parameters to minimize the transmission cost for the users without lowering accuracy. Without such expressions, it is difficult to evaluate the impact on accuracy if, for example, transmission cost is reduced without running costly iterations. Further, to keep transmission costs to an absolute minimum, our HCMS algorithm can obtain accurate counts when each user sends only a single privatized bit. We believe that our paper is one of the first to demonstrate the successful deployment of local differential privacy, [7], in a real-world setting across multiple use cases. We have shown that we could find popular abbreviations and slang words typed, popular emojis, popular health data types while satisfying local differential privacy. Further, we can identify websites that are consuming too much energy and memory, and websites where users want Auto-play. This information has been used to improve features for the benefit of the user experience.

We hope that our work will serve a role in bridging the gap between theory and practice of private systems. We also believe our work will continue to support research in a broad set of large-scale learning problems while preserving users' privacy.

References

[1] *https://www.apple.com/business/docs/iOS_Security_Guide.pdf*.

[2] C. Dwork, F. McSherry, K. Nissim, and A. Smith. **Calibrating Noise to Sensitivity in Private Data Analysis**. *TCC, 2006*.

[3] C. Dwork and A. Roth. **The Algorithmic Foundations of Differential Privacy**. *Foundations and Trends in Theoretical Computer Science, 2014*.

[4] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. **Differential Privacy Under Continual Observation**. *Proceedings of the Forty-second ACM Symposium on Theory of Computing, 2010*.

[5] G. Fanti, V. Pihur, and Ú. Erlingsson. **Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries**. *PoPETS, 2016*.

[6] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren. **Heavy Hitter Estimation Over Set-valued Data with Local Differential Privacy**. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016*.

[7] Ú. Erlingsson, V. Pihur, and A. Korolova. **RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response**. *Proceedings of the 21st ACM Conference on Computer and Communications Security, 2014*.

Contact us

Send questions or feedback ›

Jobs at Apple

Apply now ›

Tools for innovation

Apple Developer Program ›