Privacy-Preserving Crowd-Sourcing of Web Searches with Private Data Donor

Primault, Vincent; Lampos, Vasileios; Cox, Ingemar; De Cristofaro, Emiliano

Published in:
WWW '19: The Web Conference 2019

DOI:
10.1145/3308558.3313474

Publication date:
2019

Document version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Primault, V., Lampos, V., Cox, I., & De Cristofaro, E. (2019). Privacy-Preserving Crowd-Sourcing of Web Searches with Private Data Donor. In *WWW '19: The Web Conference 2019* (pp. 1487-1497). Association for Computing Machinery. https://doi.org/10.1145/3308558.3313474

Privacy-Preserving Crowd-Sourcing of Web Searches with Private Data Donor

Vincent Primault
University College London

Vasileios Lampos
University College London

Ingemar J. Cox
University College London & University of Copenhagen

Emiliano De Cristofaro
University College London

ABSTRACT

Search engines play an important role on the Web, helping users find relevant resources and answers to their questions. At the same time, search logs can also be of great utility to researchers. For instance, a number of recent research efforts have relied on them to build prediction and inference models, for applications ranging from economics and marketing to public health surveillance. However, companies rarely release search logs, also due to the related privacy issues that ensue, as they are inherently hard to anonymize. As a result, it is very difficult for researchers to have access to search data, and even if they do, they are fully dependent on the company providing them. Aiming to overcome these issues, this paper presents Private Data Donor (PDD), a decentralized and private-by-design platform providing crowd-sourced Web searches to researchers. We build on a cryptographic protocol for privacy-preserving data aggregation, and address a few practical challenges to add reliability into the system with regards to users disconnecting or stopping using the platform. We discuss how PDD can be used to build a flu monitoring model, and evaluate the impact of the privacy-preserving layer on the quality of the results. Finally, we present the implementation of our platform, as a browser extension and a server, and report on a pilot deployment with real users.

ACM Reference Format:
Vincent Primault, Vasileios Lampos, Ingemar J. Cox, and Emiliano De Cristofaro. 2019. Privacy-Preserving Crowd-Sourcing of Web Searches with Private Data Donor. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313474

1 INTRODUCTION

Over the past few years, researchers have demonstrated how online search data can be used to infer contextual information related to both individual users and populations, with examples ranging from consumer behavior modeling [19], macroeconomic statistics [14], and sales forecasting [8, 40], to early detection of cancers [30, 39] and disease monitoring at a national level [18, 21, 32].

Typically, individual-level studies rely on acquiring access to the (anonymized) search history of multiple users. However, this prompts significant ethical and privacy issues: search history often contains sensitive data on topics such as income, sexual orientation, religion, and health. Furthermore, anonymization has proven to be very difficult to guarantee [17]. A case in point is the AOL search logs, which were released in 2006: several users were easily de-anonymized, leading to a public outcry that resulted in the resignation of AOL's CTO [44]. Therefore, search engine providers seldom, if ever, provide such data, even for research purposes.

On the other hand, estimating properties of a population may be done over aggregate data, i.e., how many users have searched for specific keywords over a time frame, which is much less sensitive. In particular, in this paper, **we focus on inferring population health statistics from Web query data**, and experiment with a use-case related to estimating the prevalence of a disease such as influenza [18, 21]. Unless the population and/or region size is very small, or the queries contain many outliers, aggregates do not permit de-anonymization. One possible data source for search statistics is Google, which offers services like Google Trends and the Health Trends API (see Section 2). However, it is still quite possible that these products will not be maintained or may cease in the future. Moreover, they might not provide information on specific keywords, or sufficient geographic granularity. Also, and perhaps more importantly, since aggregation is performed by a third party, it actually provides no privacy with respect to that party.

Aiming to fill this gap, **this paper investigates how to privately crowd-source Web search data.** To this end, a number of challenges need to be addressed, including: (i) recruitment and retention, (ii) human-computer interaction, and (iii) privacy. This paper primarily focuses on a technical solution to the privacy issue. In theory, a straightforward implementation to collect users' Web search data, without the search engine's involvement, would attempt to anonymize the user and transmit data to a trusted third party, which would then aggregate data before releasing it to a researcher. However, both steps involve significant risks to individuals' privacy, with de-anonymization and security breaches of the third party being the most obvious.

By contrast, we propose a solution that does not require a fully trusted third party, and which only collects aggregate statistics on search behavior at a population level. More specifically, **we present the Private Data Donor (PDD) platform**: PDD consists of a lightweight Web browser extension, which users (data donors) install, and that monitors the searches performed by them. Users are assigned to groups, and each group uses a protocol based on homomorphic encryption to encrypt and transmit the number of times a user searched for specific queries to a server. For each user in the group, the server adds their encrypted count together. The resulting sum is an unencrypted value representing the aggregate count of the group. Data from each group is then aggregated to

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313474

determine the counts for the entire population, so that at no time is the server able to determine the query frequencies of an individual.

Contributions. Overall, we make the following contributions:

(1) We propose PDD, a new platform to handle privacy-preserving crowd-sourcing of Web search queries, decentralized, and scaling up to millions of users.
(2) We analyze the impact of different system parameters on the behavior of our platform, aiming to find the best balance between the privacy of users and the utility of collected results.
(3) We study how our platform can be applied to a real-life use case, i.e., building a flu monitoring model.
(4) We implement and deploy our platform, and use it for real-life data collection.

2 RELATED WORK

Studies using Web searches. Search engines such as Google, Bing, DuckDuckGo, etc. respond to billions of queries every day [37]. Users ask search engines a wide range of questions, and these often include personal and contextually rich information. In 2016, Google reported that at least 1% of their searches were related to symptoms, and started to provide structured information about the health conditions associated to those symptoms [31]. Overall, Web searches can often be a treasure trove for researchers, with notable examples including the applications mentioned earlier [8, 14, 19, 40].

Of particular interest to us is research relying on search data for health purposes [18, 21, 30, 32, 39], and more specifically based on crowd-sourced information [22, 36, 38, 42]. In this paper, we essentially aim to introduce a privacy-preserving layer on top of those studies, which might encourage users to donate their data, and help researchers comply with regulations, while removing the dependence on limited statistics given by search engines.

Aggregate search data. Popular search engines do not have incentives to share search data, as this information is of commercial value and, as discussed earlier, sensitive in nature. Google provides two ways to gain insights into the searches it receives, at an aggregate level: (1) Google Trends and (2) the Health Trends API. The former is a Web interface offering a global, longitudinal view of the interest for one or more queries, provided as a normalized number between 0 and 100. The latter is a service offered to researchers, with access to the interest of search queries related to health topics (see also [23, 46]). Here, the interest is provided as a frequency, i.e., the number of times the queries were searched during a time period normalized by the total number of searches in the same period. Note that this is only computed from a subset of all searches (10%-15%), and various anonymization measures are taken to remove low-count searches. With the former, the number of times a query was issued is not made available—only some normalized measure, which does not allow evaluation of the absolute popularity of a query. Moreover, statistics are only provided at a country-level or state-level granularity. Finally, relying on these two platforms means that the researchers are directly dependent on a third-party company, whereas, we envision a decentralized scenario where researchers could be autonomous in collecting the data they need.

Privacy-preserving data aggregation. Researchers have also studied the generic problem of collecting statistics from users in a privacy-preserving manner. To this end, cryptographic schemes have been proposed for privacy-preserving data aggregation, e.g., in the context of smart grids [20], wireless sensor networks [4], and streaming data [5, 26]. Our work builds on the homomorphic encryption protocol proposed in [26], which we review in Section 3.2; however, since it was originally designed for real-time applications like counting viewers of streaming videos, we need to address a number of challenges to adapt it to our setting (see Section 3.3).

An alternative approach is to add noise to the statistics so that they do not leak information about any particular individual, thus guaranteeing Differential Privacy (DP) [12]. Typically, this approach relies on a central curator, which receives raw data, aggregates it, applies a differentially-private algorithm, and releases the protected version of the data to researchers (see for example [25, 27]). However, this central entity needs to be a fully trusted entity and as such could be vulnerable to attacks or a subpoena. Another possible setup is known as local differential privacy, where noise is applied at the user-side before sending the data, thus preventing the entity collecting data from learning anything sensitive [13]. However, this requires very large numbers of users in order for the noise to cancel out and yield reasonably accurate results [16].

Privacy-preserving crowd-sourcing. Finally, prior work specific to crowd-sourcing data from users includes systems allowing third-parties to gather Web analytics [7, 15], location statistics [33], and surveys [10]. Some also provide DP guarantees [1, 6, 13]. As opposed to this line of work, we aim to provide a distributed platform that does not rely on: (i) a central anonymization server and, (ii) a persistent connection between the clients and the server.

3 PRIVACY-PRESERVING AGGREGATION

This section presents our approach to privacy-preserving collection of aggregate data, a key component of our PDD platform. After providing a high-level overview, we formalize the underlying protocol. We then discuss how to adapt it to build a large-scale system for crowd-sourcing Web searches; finally, we present the enhanced version, which we release, open-source, for public use.

3.1 Overview

Terminology. Throughout the paper, we use the following terms:

- *Server:* in PDD, the server is a piece of software running for an institution interested in gathering Web searches statistics.
- *Users:* people willing to contribute their searches (in aggregate form) to a research project. They install a *client*, i.e., a browser extension communicating with the server. We use the terms "users" and "clients" interchangeably in the rest of the paper.
- *Query:* a string that can interpreted by a search engine (e.g., Google, Bing, DuckDuckGo) to retrieve matching Web resources, e.g., "world cup odds," "will it rain today," etc.
- *Monitored query:* a query for which the server collects and provides statistics.
- *Search:* a query that was sent by a user to a search engine. A search is thus the association of a user, a query, and a timestamp.
- *Analyst:* a researcher having access to the data collected by PDD and using it for further analyses.

Data Aggregation. We build on an underlying privacy-preserving data collection protocol – specifically, the one presented by Melis et al. [26] – to create a fully-fledged crowd-sourcing platform for Web searches. The protocol encompasses three steps:

(1) *Setup:* Users willing to contribute their searches (in aggregate form) install a browser extension; upon installation, the extension generates a pair of cryptographic keys, registers itself to the server with its public key, and is assigned a client identifier.
(2) *Reporting:* Clients regularly obtain the set of monitored queries and the public keys of the other clients. They generate a (secret) blinding factor for each query, in such a way that the sum of blinding factors of all users, for a given query, will equal zero. Using their browsing history, clients fill a vector of counts, where each value corresponds to the number of times a monitored query was searched in the previous day: the vector is encrypted using the blinding factors and sent the server.
(3) *Aggregation:* Once the server has received the encrypted vectors of all clients, it can decrypt the encrypted counts by summing them up.

Adapting Melis et al.'s protocol to our setting immediately prompts two main challenges. A first issue is the impact of a client being unable to report his encrypted data to the server; in this case, the sum of all blinding factors is *not* equal to zero, and the encrypted data cannot be decrypted by the server. In other words, a single user failing to report his data on a given day will result in the inability to retrieve the statistics for a particular day, which is not acceptable. Our PDD platform introduces two measures to build more reliability into the protocol: (1) We organize clients into smaller groups, each one running its own instance of the protocol; (2) We introduce some additional delay before "giving up," leaving more time for clients to send their data.

Another issue is that [26] requires a persistent connection between the server and the clients. This allows the server to learn when users are online, which may introduce a new privacy threat, and also increases the load on the server. We remove this requirement by letting clients communicate with the server at most once a day, which significantly lowers the privacy threat.

3.2 Privacy-Preserving Aggregation

We now formally present Melis et al.'s protocol [26].

Notation. First, let us introduce notation, also summarized in Table 1. We use N to denote the numbers of clients in the system, and L the number of monitored queries. $C_i, i \in [1, N]$ denotes the i-th client, and $W_j, j \in [1, L]$ denotes the j-th monitored query. Let $\mathbb{G}$ be a cyclic group of prime order q for which the Computational Diffie-Hellman problem [11] is hard, and g be the generator of the same group. Also, we use a cryptographic hash function, $H : 0, 1^* \to \mathbb{Z}_q$, mapping strings of arbitrary length to integers. Let "$||$" denote the string concatenation operator, and $a \in_r A$ indicate that a is sampled at random for A.

Algorithms. The protocol involves the following algorithms:

- *Initialization.* Each client C_i generates a private key $x_i \in_r \mathbb{G}$ and a public key $y_i = g^{x_i} \bmod q$, and then sends its public key along with its identifier i to the PDD server to be registered.
- *Encryption.* At each round s, client C_i holds a counts vector $V_{si} = (v_j \in \mathbb{N}, j \in [1, L])$, where v_j represents the number of times the query W_j was searched for. To encrypt this vector, client C_i first generates a blinding factors vector:

Symbol	Meaning
$\mathbb{G}$	Cyclic group of prime order q, whose generator is g
H	Hash function mapping strings to integers
$\vert\vert$	String concatenation operator
s	Current round number (0-based)
N	Total number of clients
C	Set of all clients
C_i	i-th client, $i \in [1, N]$
(x_i, y_i)	Private and public keys of client C_i
L	Total number of monitored queries
W_ℓ	ℓ-th monitored query, $j \in [1, L]$
K_{si}	Blinding factors of client C_i at round s
V_{si}	True counts vector of client C_i at round s
$\hat{V}_{si}$	Encrypted counts vector of client C_i at round s
$o_s : C \to \mathbb{B}$	Whether client C_i was online during round s

Table 1: Notation.

$$K_{si} = \left(\sum_{j=1, j\neq i}^{N} H(y_j^{x_i} || \ell || s) \times (-1)^{i>j} \bmod q, \ell \in [1, L] \right) \quad (1)$$

where $(-1)^{i>j} = -1$ for $i > j$ and 1 otherwise. Then, each C_i encrypts its counts vector V_{si} into an encrypted vector $\hat{V}_{si} = V_{si} + K_{si}$, which is then to be sent to the server.

- *Decryption.* Once the server has received the encrypted vectors for all clients at round s, it can decrypt it by summing them:

$$V_s = \sum_{i=1}^{N} \hat{V}_{si} = \sum_{i=1}^{N} V_{si} + \sum_{i=1}^{N} K_{si} = \sum_{i=1}^{N} V_{si} \,. \quad (2)$$

Indeed, the sum of all K_{si} is a zero vector:

$$\forall \ell \in [1, L], \sum_{i=1}^{N} K_{si_\ell} = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j\neq i}}^{N} H(y_j^{x_i} || \ell || s) \times (-1)^{i>j} = 0 \,. \quad (3)$$

3.3 Challenges

As mentioned above, using Melis et al.'s protocol [26] in our setting (i.e., crowd-sourcing Web searches) poses a few challenges, as it was designed for a different set of applications, namely, counting the viewers of streaming videos or the number of requests to Tor Hidden Service directories. In other words, [26] targets real-time data collection in a context where users are expected to all be online at the same time and the cryptographic code actually runs directly in the browser in Javascript (specifically, as a Node.js package). At some point, the server decides to gather statistics, issues a request to the clients to encrypt and send their counts, and decrypts/publishes the aggregate results after decrypting the clients' contributions.

However, if some clients disconnect during the process, e.g., the user goes offline or closes the browser tab, the server will not receive some of the encrypted counts, and cannot decrypt the aggregate counts since the blinding factors no longer sum up to zero. Also, the protocol requires a persistent connection between clients and server (e.g., via a web-socket or a regular ping). We discuss these two challenges in more detail next.

Disconnections. To mitigate the issue of disconnections, [26] proposes two solutions: (1) organizing users in smaller groups, and (2) running a recovery protocol. Specifically, the protocol can be applied independently on several groups, and the decrypted count vectors of each group summed to obtain the global decrypted count vector. In this setting, the size of each group becomes a critical parameter, impacting both privacy (i.e., among how many other users is each user hidden) and utility (i.e., the likelihood the group's count vector can be decrypted). Put simply, the larger the group, the better the privacy, but the more at risk is utility. Another open question, which was not handled in [26], is how to organize users into groups. At the same time, the recovery protocol allows the users that are still online to compute a recovery factor, which will then be used to "reverse" the effect of the blinding factors of missing users. However, users might in turn disconnect during this recovery protocol, making the recovery protocol fail in turn.

Persistent connection. In our setting, the need for a permanent bi-directional connection between the server and its clients is very undesirable, for two main reasons. First, having a permanent connection between the server and every client requires non-negligible computing resources, while we need the platform to be as operationally simple as possible, allowing researchers to deploy it on a single server with limited computing power. Second, this also raises orthogonal privacy issues, as the server will know the times at which users are online, learning information about users' behavior which can potentially to be a side-channel for privacy attacks. In theory, privacy-preserving presence protocols have been proposed (e.g., [3]), but they rely on having multiple non-colluding servers, which we do not have.

Requirements. Overall, as mentioned, Melis et al.'s protocol [26] was designed for different applications, while we are interested in collecting Web searches. This distinction has two practical implications in the requirements of our platform: (1) Our client-side platform needs to access browsing history to collect the search queries (specifically, the monitored queries), thus, it needs to run as a browser extension (and not directly on a particular web page); (2) We do not need real-time statistics, but rather daily aggregates.

Next, we discuss how we modify the platform to accommodate our requirements and overcome these challenges.

3.4 Enhanced Aggregation Protocol

The main idea is that the server prepares itself at the beginning of every day to gather statistics about the past day. To do so, it pre-computes, each night, groups of clients based on their past activity patterns. Then, throughout the day, when they come online, clients ask the server which groups they are part of, and retrieve the list of monitored queries. Each user is only part of at most one group each day, but the server may return several groups for the past few days, if necessary to recover from past days where the user was inactive. Clients then use their browsing history to compute the counts vector for each requested group, encrypt it using the public keys of the other members of the group, and send the encrypted counts vectors to the server. In this setup, the round number s becomes the number of days elapsed since the data collection began.

Handling disconnections. As proposed in [26], we organize clients into groups. However, as we only have at most one contact with each client per day, the challenge is to be able to pre-compute groups without knowing for sure which users will be active the next 24 hours. To this end, we leverage historical knowledge about past clients' activity. More specifically, at the beginning of day d, when statistics about day $d-1$ need to be gathered, the server knows which clients were active during day $d-1$. Therefore, the server includes only the clients active on day $d-1$ in the groups formed on day d, which reduces the number of clients actually involved in the protocol every day, and improves the reliability of the data collection. Indeed, the less users are involved in the protocol, the lower the probability that some of them will not be active on day d and hence the lower the probability to lose data.

Creating groups introduces, obviously, the need for setting a *group size*, which is a system parameter controlling the number of clients into each group. Concretely, groups formed at the beginning of day d are composed of only clients that were active on day $d-1$, and affected to groups in a round-robin fashion. All groups will hence have the same number of clients, except one group which may contain less clients. We will explore, in Section 4.4, a more advanced strategy, selecting users by their past activity, trying to maximize the probability for them to be online.

Nonetheless, the main drawback of pre-computing groups is that there is still a risk that some clients involved in the protocol will not be active the next day, which may prevent the encrypted data from some groups to be decrypted. For that reason, we introduce the notion of a *delay*, i.e., a number of days during which the server will wait for the encrypted counts to be sent by the clients. Waiting a few more days will allow the inactive clients to come online later on, thus improving the quality of the results.

Removing persistent connections. Because clients only contact the server at most once a day, the privacy threats are dramatically lowered as compared to having a persistent connection, because the server does not anymore know, in real-time, which users are active. It also significantly lowers the computational pressure on the server, which is only expected to handle a couple of requests from each client every day. (Rate limiting strategies should also be used to protect against DDoS attacks [43].)

Confidence. It might still happen that a client does not come online on day d, and neither in the additional delay that may be granted. In that case, it means that the encrypted data for this client, and for all this group, will never be decrypted and is considered as *lost*. Although lost data should be considered impossible to recover at that point, we can provide a *confidence* measure to the data analysts, computed as the ratio of successfully decrypted users. This adds transparency to the figures provided to them by the PDD platform.

Threat Model. The security of this approach relies on that of the aggregation scheme presented by Melis et al. [26], which is proven secure in the honest-but-curious (HbC) model under the Computational Diffie-Hellman assumption [2]. Here the HbC model means that both the clients and the server, although they may attempt to violate privacy of their counterparts, do not deviate from the protocol specification. For instance, malicious users could report fake values in order to invalidate the final aggregation values, while the server could maliciously manipulate the number and identity of clients in the same group. This is a common assumption in cryptography, which can be mitigated using verifiable tools such as zero-knowledge proofs, cryptographic commitments, and code attestation. We leave such extensions as part of future work, as they are orthogonal to the challenges we face in this work.

3.5 Example

To clarify how our platform works, we use an example, as pictured in Figure 1. We consider five clients (A, . . ., E), who participate in

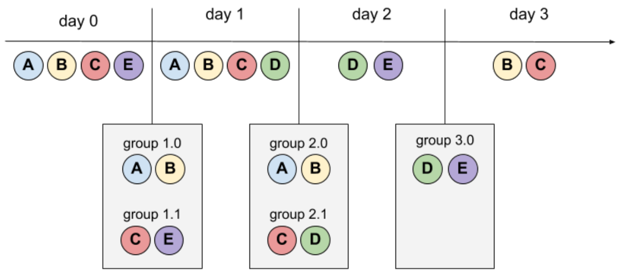


Figure 1: Example of how users can be added to groups based on their past activity.

data collection, a group size of two, and a delay of one additional day. At the beginning of day 1, A, B, C and E, who were seen online on day 0, are put into two groups of two clients each. On day 1, A, B and C were seen online, as well as client D, who is not part of any group. Then, A, B and C will be tasked with sending their encrypted count vectors to the server, while D will not have any task to perform. At the end of day 1, data of group 1.0 has been decrypted, but data of group 1.1 is still missing E's counts vector.

At the beginning of day 2, A, B, C, D (who were seen online on day 1) are put into two groups. At the end of day 2, because E has eventually come online within the one additional day of delay, data of group 1.1 has been decrypted. However, groups 2.0 and 2.1 are still un-decrypted, because A, B and C were not online on day 2. At the end of day 3, group 2.1 has been decrypted within the one-day delay, but group 2.0 is lost since A was not active on days 2 and 3.

Eventually, data for day 0 has been made available at the end of day 2 (with a confidence of 100%), while data for day 1 has been made available at the end of day 3 (with a confidence of 50%). Overall, this simple example illustrates the usefulness of both dividing users into groups (allowing to decrypt half of the data for day 1) and the additional delay (allowing to decrypt all, instead of only half, of data for day 0).

4 CONFIGURING PDD THROUGH SIMULATION

We use extensive simulations to determine the configuration of the PDD platform. Specifically, we evaluate the impact of the two main system parameters (see Section 3.4): the group size and the delay. As PDD is a distributed system, potentially involving thousands or even millions of users, we use simulation to better understand its behavior and test different parameterizations.

4.1 The PDD Simulator

We now introduce the simulation environment used in our experiments. It involves two distinct modules, simulating: (1) user activity, and (2) the PDD protocol itself. The former simulates how users interact with their browser, while the latter simulates the interaction between clients and the server. The simulator is implemented in Python (for data pre-processing and analysis) and Scala (for the main workflow).

User activity module. The user activity module is made of three distinct parts, each simulating: (i) connectivity, i.e., which days

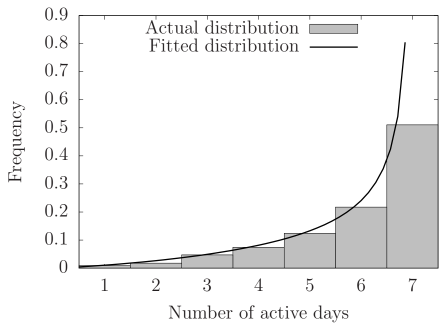


Figure 2: Distribution of users' activity.

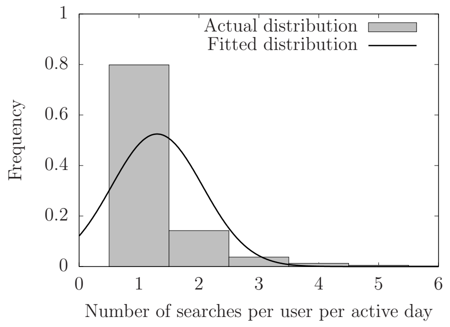


Figure 3: Distribution of search volume.

users are active and using their browser, (ii) search volume, i.e., how many searches users are performing on active days, and (iii) search content, i.e., what are users actually looking for.

Overall, accurately simulating users' browsing activity is a very challenging task, mainly due to the lack of publicly available datasets. To overcome this challenge, we turn to the datasets released by Mozilla in 2010 to shed light on how users interact with their browsers [28]. We use the most complete dataset, which contains data for 4,802 users, collected over a week in December 2009. For each user, we have the number of browsing sessions (i.e., how many times each user started his browser during the week) and the average duration of those browsing sessions. We use the data to "replay" such sessions since, unfortunately, the dataset does not contain the *times* at which users were active. We obtain a distribution with the number of days (out of seven) during which the users have been effectively using their browser. Due to the randomness of this process, we run 100 iterations, and eventually select the Beta distribution with the lowest distance, according to a Kolmogoroc-Smirnov test [24]. More specifically, we end up with a Beta distribution with $\alpha = 2.2170$ and $\beta = 0.4634$.

The original frequencies and the fitted distribution are shown in Figure 2. Note that the fitted distribution has a mean of 82.7% – i.e., on average, users are using their browsers between 5 and 6 days every week. This is consistent with a more recent study from Mozilla [9], which reports how long Mozilla's data scientists have to wait for telemetry data to be available, estimating that, on average, 95% of telemetry data was received within a single day. For each user, we sample a number of active days from the Beta distribution, scaled to the total number of days considered in our experiments (rounded to the closest integer). Simulations in Section 4 involve 100k simulated users over 100 days.

Next, we simulate search volume using the Yahoo dataset [35], which provides search logs collected over a month (July 2010), featuring 29M users, 67k unique queries and 80M searches. From this dataset, we extract the daily number of queries performed by each user, and fit a normal distribution on it ($\mu = 1.3020$, $\sigma^2 = 0.7603$). Figure 3 reports the original data along with the fitted distribution. For every day a user is active, we sample a number of searches to be performed from the distribution (rounded to the closest integer). Finally, we simulate searches by extracting the frequency of each of query included in Yahoo's dataset, which spans over an entire month. Note that the data is truncated after more than 10k searches of a given query, which may limit the precision of the simulation for very frequent queries. However, it still allows us to capture that a few queries are searched very often (e.g., 2k queries more than 10k times), with a long tail of infrequent queries (e.g., 11k queries less than 10 times). We simulate searches by randomly sampling a query from this distribution, as a random choice weighted by the frequency of each query.

PDD protocol module. This module trivially simulates the behavior of the PDD server by implementing the protocol described in Section 3.4. We assume that network communication is reliable (using TCP), thus, we do not need to simulate network communications and failures. Because we rely on simulation instead of a real-life system, we also have access to the ground-truth data (i.e., the V_{si}'s, the counts vectors of each client at each round), which we can use to evaluate our solution. The simulation essentially produces as output, for every day, a report about the true count and estimated count (i.e., after decrypting data) of each query, the total number of searches and the confidence.

Evaluation metric. We use the mean relative error (MRE) of the decrypted counts as the main metric to evaluate the quality of the collected data, averaged out across all queries, all users, and all rounds. Formally, it is defined as:

$$\text{MRE} = \frac{1}{|\{V_{si_\ell} \neq 0\}|} \sum_{V_{si_\ell} \neq 0} \frac{V_{si_\ell} - \tilde{V}_{si_\ell}}{V_{si_\ell}}, \quad (4)$$

where $\tilde{V}_{si}$ is the estimated counts vector received by the server for client C_i at round s. In practice, $\tilde{V}_{si} = V_{si}$ if the data of client C_i was successfully decrypted at round s, or $\tilde{V}_{si} = (0, \ldots, 0)$ otherwise.

4.2 Group Size

We start by using simulations to study the impact of the group size. First, users are put into groups following the heuristic described in Section 3.4, considering only the users who were online the previous day, i.e., at day d, $\{C_i \in C | o_{d-1}(C_i)\}$.

Group size is inherently responsible for the trade-off between privacy and utility. Here the level of privacy provided to the users is akin to k-anonymity [12], meaning that every client part of an aggregate will be hidden among $k-1$ other clients being part of the same aggregate, whereas, utility is measured via MRE as discussed above. On the one hand, creating groups of very few clients would yield near-optimal utility, but at the cost of providing little or no privacy to the users. On the other hand, putting *all* clients into a single group would provide the best levels of privacy, but can dramatically decrease the utility, since even a single client not sending its data will prevent the decryption of everybody's data. It also means that system designers have no control on the impact that increasing/decreasing numbers of clients will have.

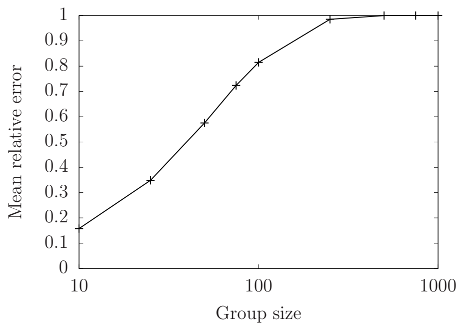


Figure 4: Impact of the group size on utility.

Therefore, we seek to find a middle ground, providing reasonable privacy guarantees, while still allowing reported counts to be accurate enough for data analysts to work with it. In Figure 4, we report the impact of the group size on the MRE of decrypted counts with a delay of two days. We report results with group sizes between 10 and 1000, as suggested in [26]. We note that MRE varies between 16% and 100%, increasing quite quickly: a group size of 100 already yields an 81% MRE. The privacy-utility trade-off is pretty clear: groups should not be too small because of privacy (in fact, we would not recommend less than 50 clients per group), however, large group sizes do not seem very practical, e.g., MRE is 100% with 750 users per group. Therefore, in Sections 4.3–4.5, we will experiment with group sizes between 50 and 500, and study how we can improve the utility without degrading privacy.

4.3 Delay

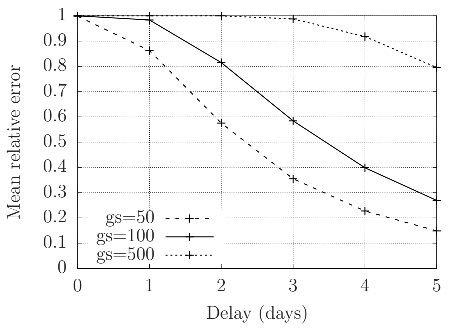
Next, we study the impact of utility when waiting for a few more days to provide aggregate counts to the analysts. As opposed to the group size, the delay does not play a central role in the privacy-utility trade-off, as it does not impact the privacy. Instead, it imposes a trade-off between utility and the availability of the results. Figure 5a shows the evolution of MRE with delay varying between 0 and 5 days, for group size $gs \in \{50, 100, 500\}$. We note a significant positive impact on MRE starting with a delay of two days.

Choosing the right delay is inherently application-dependent: some use-cases might accommodate well delays of a week, while others need near-real-time results. Nevertheless, Figure 5a shows that a group size of 500 is still not very practicable, because the MRE remains around 80% in the best case, while groups of 50 clients with 5 days of delay reduces MRE to 15%.

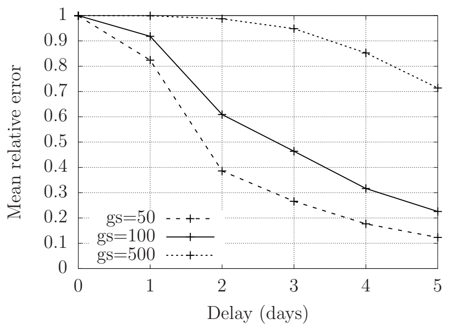
Our simulations also highlight that some errors are not recoverable, even while waiting up to five additional days. We believe this is due to the behavior of users, as shown in Figure 2, with a very small fraction of the users being active only once in a while. Next, we study how to mitigate the impact of inactive users on MRE.

4.4 Improving Client Selection

Aiming to improve the privacy-utility trade-off presented above, we present a novel approach to organize clients in groups. The basic idea is to only put into groups clients that are likely to be active the next few days, assuming that some additional delay is allowed.



(a) Naive approach.



(b) Improved selection algorithm.

Figure 5: Impact of the delay on utility.

Indeed, we want to maximize the probability that all users inside groups will be active in the next days, even if it implies losing out on the data of some users that are generally not very active.

Ideally, one would rely on machine learning techniques to learn user behavior patterns. However, because of the lack of real-life browser usage traces, this approach is not viable since we would not be able to effectively train prediction models. Therefore, we set to improve our heuristic to select users to put into groups more effectively. As shown in Figure 2, while most users are active every day, there is a non-negligible tail of users that are much less active.

Basically, if we have a delay of δ days, we want to only select users that are active on average at least one out of δ days. More formally, at the beginning of day d, when the delay is configured to δ days, we only select users satisfying the following constraint:

$$\left\{ C_i \in C | o_{d-1}(C_i) \wedge \frac{|\{d' \in [0, d-2], o_{d'}(C_i)\}|}{d-1} \geq \frac{1}{\delta} \right\} . \quad (5)$$

The above criterion is applied when $\delta > 0$ and $d > \delta$, meaning that we effectively have some additional delay and have accumulated enough data about users' past behavior to decide whether it is worth including them into a group.

Figure 5b shows the impact of this new heuristic on MRE. When compared to Figure 5a, we observe a noticeable decrease in the error. For instance, with a group size of 100 and 2 days of delay, the MRE goes from 0.82 to 0.60 (a 26% improvement). In fact, the best improvements with the heuristic are obtained with 2 days of delay. With 1 day of delay, we do not have enough past activity history to yield enough impact, while further increasing the delay is sufficient to lower MRE without the need for our heuristic.

4.5 Building Confidence in the Platform

Finally, we set to provide a confidence metric to the analysts in order to capture the expected quality of the aggregate statistics. Essentially, we use the percentage of users that were successfully involved in the data collection process. That is, we define confidence as the ratio of clients whose data was successfully decrypted and the total number of clients who were active on the target day. For example, a confidence of 60% for day d means that we were able to decrypt data for 60% out of all clients who were active that day. Recall that the missing counts can be due to clients that never sent their encrypted counts to the server, or by clients who were part of a group where a client never sent their data.

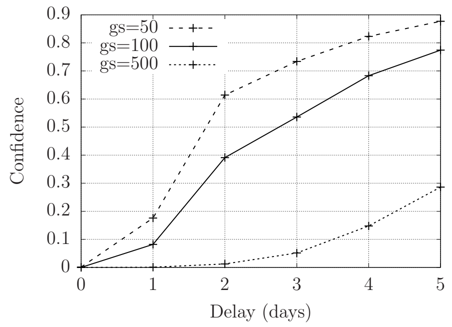


Figure 6: Confidence levels, i.e., ratio of clients whose data was successfully decrypted, for increasing delay.

In Figure 6, we plot the evolution of the confidence depending on the delay and the group size. Unsurprisingly, this is closely related to MRE (cf. Figure 5b), i.e., a high MRE is associated to low confidence. However, while MRE can only be computed through simulation (since it relies on the ground truth), confidence can actually be computed in real production environments. Therefore, we believe it can constitute a simple yet effective tool for analysts to understand the accuracy of the statistics at their disposal. However, it is not so easy to understand the actual impact of MRE, and what is the threshold differentiating between a "good enough" and a "bad" MRE. In the next section, we will experiment with the PDD platform on a real-life use-case, and study the actual impact of the privacy "layer" on the accuracy of the results obtained by analysts.

5 BUILDING A SYNDROMIC SURVEILLANCE MODEL FOR INFLUENZA-LIKE ILLNESS

In this section, we experiment with a health-related use-case, showing how data collected through PDD can be used to monitor the prevalence of influenza-like illness (ILI). By using an established model for this task that provides a significant level of accuracy [21, 22], we are particularly interested in quantifying the impact that introducing PDD may have on its performance.

Flu season	**True query frequencies**			**Estimated query frequencies**		
	MAE	RMSE	*r*	MAE	RMSE	*r*
2014/15	2.5211	3.7022	0.8631	3.0163 (0.6112)	4.0382 (0.6251)	0.8574 (0.0211)
2015/16	2.1023	2.8766	0.9439	2.4415 (0.1949)	3.1805 (0.2586)	0.9202 (0.0140)
2016/17	2.1907	3.3163	0.9165	2.0212 (0.2980)	2.6704 (0.3501)	0.9020 (0.0313)
Avg.	2.2714	3.2984	0.9078	2.4930	3.2964	0.8932

Table 2: Performance of the ILI rate estimation task

5.1 Methodology

Starting from the simulator described in Section 4.1, we modified the users' activity module to sample the searches from a set of actual flu-related queries. More specifically, we use data from Google Health Trends to accurately model the behavior of users when interacting with the Google search engine. We retrieve the daily frequencies of 1k flu-related queries from January 1, 2008 to December 31, 2017 (10 years). We follow the approach presented in [23] to extract and filter queries based on their semantic relationship with the topic of flu. Note that we only have access to frequencies (and not absolute counts), i.e., the number of times a query was issued over the total number of queries issued that day. For every query, we fit a frequency distribution using a maximum likelihood estimation. We split the queries in two classes: sparse queries, which are fitted to a beta distribution, and denser queries, fitted to a generalized extreme value distribution. During the simulation, we use those distributions to sample, for each query and every day, a frequency.

In total, the added frequencies of the 1k queries of interest account on average for about 0.34% of the daily number of searches. This means that, as opposed to the previous behavior, the large majority of the searches that are generated are not of interest for influenza modeling. Consequently, we run a much larger-scale simulation in order to collect enough search samples, even for sparse queries; to this end, we simulate 1 million users over 20 years, which yields 825k samples for the 1k queries of interest. We use a delay of 2 days and a group size of 100 clients.

We then use the outputs of the simulation to compute daily query frequency estimates, which are then compared with their corresponding true values. We use a relative error metric, i.e., $(\mathrm{f_t} - \mathrm{f_e})/\mathrm{f_t}$, where $\mathrm{f_t}$, $\mathrm{f_e}$, denote the true and the estimated query frequency, respectively. Note that, on the frequency space, errors may be positive as well as negative. Given that positive and negative errors have different properties, we model them separately using two Weibull distributions per query (different distribution parameters for each query). Moreover, we capture the overall tendency for positive versus negative errors during daily data extracts, modeled by a Gaussian distribution. We filter out queries with less than a 100 nonzero true frequency estimates from this process, as their error samples are very sparse (466 from 1k queries were kept).

Finally, we use the fitted error distributions to introduce errors to true query frequencies from Google Health Trends. We rely on a two-step sampling process, where first the error signs are drawn for all queries for particular day, and then for each query, based on its allocated error sign, a relative error is drawn from its positive or negative error distribution. We deploy 50 sign draws per day, and for each one 50 error distribution draws per query, which generates 2,500 query frequency data sets where errors have been propagated.

5.2 Evaluating the Impact of PDD

Next, we set to assess the effect of the errors discussed above in the context of a practical task. To this end, we used the estimated query frequency datasets as inputs for modeling ILI rates in England – a well-studied task in the literature [18, 21, 23, 41].

Model structure. Disease rate estimation from online search data is commonly formulated as a regression task [18, 21]. The aim is to learn a function $g\colon \mathbf{X} \to \mathbf{y}$ that maps the input space of search query frequencies, $\mathbf{X} \in \mathbb{R}^{n \times m}$, to the target variable, $\mathbf{y} \in \mathbb{R}^{n}$, representing disease rates; n denotes the number of samples and m is the number of unique search queries we are considering. The time interval for each sample is set to a week to match the frequency of syndromic surveillance reports. Here, we use elastic net [47] as our regression function, similarly to previous work on the topic [21, 22]. Elastic net combines ℓ_1-norm and ℓ_2-norm regularization to obtain sparse solutions and to address model consistency problems that arise when collinear predictors exist in the input space [45]. It is formulated by the following optimization problem:

$$\underset{\mathbf{w},\beta}{\operatorname{argmin}} \left(\|\mathbf{y} - \mathbf{X}\mathbf{w} - \beta\|_2^2 + \lambda_1 \|\mathbf{w}\|_2^2 + \lambda_2 \|\mathbf{w}\|_1 \right) , \tag{6}$$

where $\lambda_1 > 0$, $\lambda_2 > 0$ are respectively the ℓ_1-norm and ℓ_2-norm regularization parameters, and β denotes the intercept term.

Metrics. We assess the performance of elastic net models in estimating ILI rates in England during three flu seasons (2014–2017). Test periods are 1-year long (52 weeks) and training sets include all data prior to the corresponding test period. Prior to applying elastic net, we filter out queries with a Pearson correlation < 0.3 with the ground truth (per training set). We measure performance using mean absolute error (MAE), root mean squared error (RMSE), and Pearson correlation (r).

Results. We report performance figures in Table 2. Overall, we see that the average performance (based on the 2,500 trials) obtained by the estimated query frequency data does not differ much to the one obtained by the error-free data. There is a 9.7% increase in MAE, a 1.6% decrease in correlation, while RMSE remains unaffected. Figure 7 depicts ILI rate estimates across all the test periods in comparison to ground truth data. We plot the mean ILI rate across the 2,500 trials and use two standard deviations as a confidence interval. There are minor differences in the estimates of the privacy-preserving and true frequency models, although overall the time series appear as visually similar.

Overall, our analysis shows that both provide good estimates of the actual ILI rates, which serves as a practical indication that the error introduced by the privacy-preserving scheme does not severely affect the utility of the search data.

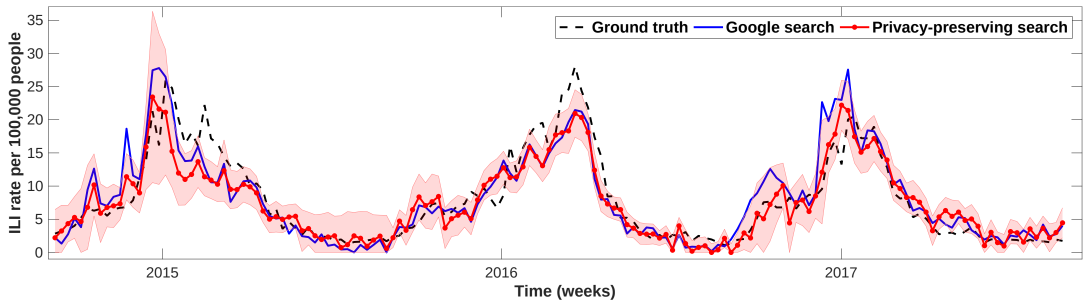


Figure 7: ILI rate estimates based on true and estimated (privacy-preserving) query frequency values compared to the ground truth. Confidence intervals depict two standard deviations from the mean for the 2,500 drawn query frequency estimates.

6 REAL-WORLD DEPLOYMENT

As mentioned earlier, we have implemented PDD as a fully-functional platform, which is freely available to health researchers. In this section, we detail our prototype, presenting its architecture as well as a real-world experiment with it.

6.1 System Architecture

PDD follows a classical client-server architecture, as depicted in Figure 8. The client-side is deployed to each user as a browser extension for Google Chrome/Chromium. The server is hosted by the research institution in charge of running the data collection campaign. A Web dashboard is also available to administrators and data analysts, allowing them to tune the data collection parameters and to export the results once they are available.

Server. The PDD server is the central component of our architecture, implementing three core services. First, it provides the concept of a *campaign*, which corresponds to a set of queries being monitored over some period of time. Several campaigns can be running independently in parallel, allowing different research teams to rely on a common crowd-sourcing infrastructure for their research. Second, the server is in charge of regularly initializing empty count vectors. Once a day, it scans the active campaigns and clients and organizes the latter into groups according to the algorithms described in Section 4. One or several empty count vectors are then initialized for each client that is expected to contribute some data to one or more campaigns. Each time a client pings the server, the latter will look for all those yet-to-be-filled vectors, and asks the client for the missing data. Third, the server is responsible for the aggregation of count vectors. Once a day, this service collects the encrypted vectors sent by the clients, decrypts them whenever possible, and updates the global count vector for the target day.

Each campaign has its own system parameters (i.e., delay and group size) as well as a distinct list of queries of interest. Note that, while Section 3 only describes the ability of monitoring queries, our implementation can monitor both keywords as well as queries. That is, by splitting every search around whitespaces, we can extract tokens and monitor the searches that contain them or combinations of them. For instance, instead of monitoring the query "symptoms of flu," an analyst might choose to track the keywords "symptoms" and "flu," allowing to match different variations such as "flu symptoms."

Communication between the clients and the server is through a REST API, secured by TLS. The REST API itself is divided in

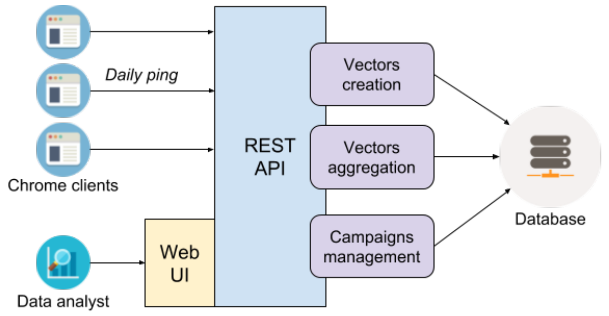


Figure 8: Architecture of the PDD system.

two parts: a public and a private APIs. The former is accessible to the clients and essentially contains three endpoints: (i) client registration, used when a Chrome extension is installed, (ii) ping, used by clients to get their instructions, and (iii) vector submission, used to update the vectors back as instructed by the server. The latter is only accessible to the analysts and is used for administrative operations. The server is written in Scala (a language running on the Java Virtual Machine), while the dashboard in Javascript. The data is stored in a MySQL database, although the persistence layer is designed to be pluggable to accommodate other storage systems.

Client. The client-side is implemented as a Google Chrome extension, although it can be ported to other browsers, such as Firefox, with minimal effort. The extension includes 733 source lines of code written in Javascript. The main goal of the extension is to contact the server on a daily basis (whenever possible) via a ping. The response of a ping is a set of instructions about some information to compute in order to fill empty count vectors. Each instruction contains a set of monitored queries, the period of interest (typically a day), the round number and the public keys of the members of the group. The client is then in charge of computing the count vectors, thanks to the API provided by Chrome, by locating the Google searches from their URL, extracting the query, tokenizing it into keywords and aggregating them with respect to the set of monitored keywords. This vector is then encrypted, using the public keys provided and round number, and sent back to the server.

The browser extension also has a lightweight interface for the user, providing transparency about the collection process. Notably, it allows the user to see which searches are going to be sent to the PDD server at the beginning of the next day. If the user does not want some of this data to be contributed (even though it is aggregated and encrypted), he has the option of creating a blacklist.

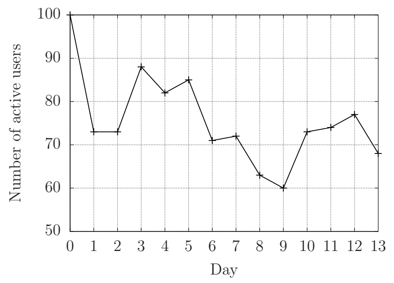


Figure 9: Daily activity during the real-life experiment.

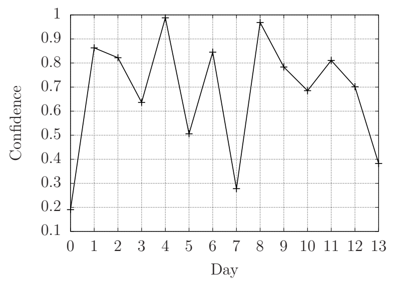


Figure 10: Daily confidence during the real-life experiment.

The latter contains a list of keywords that the user does not want to be sent to the server, permanently.

Source Code. The PDD platform is available in open source, under the GNU General Public License v3, on GitHub.[1] The Chrome extension is also available on the Chrome Web Store.[2]

6.2 Real-World Experiment

In order to experiment with our prototype and real-life users, we used the Prolific.ac [29] website to recruit a pool of (paid) volunteers. Our study received ethics approval from University College London. In total, we ran two studies. The first one aimed to validate that the system was actually working and reliably collecting data, and involved 10 participants for a week. A couple of months later, we launched a larger-scale study involving 100 participants over two weeks. In the following, we report on the latter.

Participants were asked to install our Chrome extension and leave it active, but were not otherwise asked to alter their behavior in any way. We only required them to be active for at least 10 days (out of 14 in total), in order to prevent dishonest participants claiming a reward without actually contributing any data. Because the season was not favorable to experiment with flu-related searches (it was done during summer 2018), we instead monitored the usage of the 10k most common English words.[3]

User behavior. To analyze the activity of the participants, in Figure 9, we show the number of daily active users throughout our experiment. The plot highlights that there are indeed variations in the number of daily active users, varying between 60% and 90%. This also confirms that we cannot realistically expect, even with as little as 100 paid users, that all users will be active on a single day. In fact, on average, users were active 10 days out of 14. Overall, we detected a total of 3000 to 5000 keywords per day. This rather high number is due to the fact that we are monitoring the 10k most common English words, meaning that each search will likely generate several keywords, one for each of the common words in it.

Error. Because we had a small number of users, we used a group size of only 10, fixing the delay to 2 days. Figure 10 plots the confidence throughout the 14 days of our experiment. The average confidence is 71.3%, even though there are significant variations. Note that the results for the first day exhibit very low confidence, because of the participants who left the experiment and were never active again. The confidence for day 7 is also low at 28%; it corresponds to a turning point between both a peak of active users at day 5 and the lowest number of active users at day 9 (cf. Figure 9), meaning the users' future activity was overestimated when creating groups for day 7.

Performance. The PDD server is packaged as a Docker image and deployed on a single virtual machine equipped with 4×2.1 GHz and 16 GB of RAM. Over the 2 weeks, the user CPU usage was on average 0.2%, while the memory consumption was on average 2.1GB. We did not experience any crash, and the resource usage leaves plenty of room to grow to more users.

7 CONCLUSION

This paper introduced PDD, a platform enabling researchers to access crowd-sourced Web searches, collected in a privacy-friendly way. We built on an existing cryptographic protocol for privacy-preserving data aggregation [26], adapted it to our settings, and evaluated our solution through extensive simulations. In particular, we highlighted the importance of two parameters, delay (i.e., the number of days to wait for the results) and group size (i.e., the anonymity of users). We then validated the suitability of our approach on a real-life use case, building a model monitoring the prevalence of influenza from Web searches. We compared the quality of the model produced when using PDD to both ground truth and the model obtained from *raw* Google searches, and observed very similar performance. Finally, we deployed our PDD prototype in the wild and ran a small pilot deployment involving 100 paid participants, achieving a reasonable confidence (more than 70%) while keeping resource requirements low.

Although we focus on health-related use cases, we believe that PDD can accommodate other contexts. Overall, we envision the opportunity to build a fully-fledged privacy-preserving analytics platform on top of this work, paving the way to more use cases while enforcing strict privacy guarantees. Our experimental evaluation showed opportunities to improve even more the way we select and organize users into groups; we believe that leveraging machine learning, in addition to our heuristics, could greatly increase the confidence while lowering error. In future work, we also plan to study the feasibility of possible inference attacks against aggregate statistics (e.g., membership [34]), and mitigate them using differentially private mechanisms.

Acknowledgments. This work has been supported by the EPSRC grants EP/K031953/1 and EP/R00529X/1. We also wish to thank Google for providing access to the Health Trends API and Lukasz Olejnik for early work on the PDD prototype.

[1]https://github.com/pddisense/pdd
[2]https://chrome.google.com/webstore/detail/private-data-donor/ipeekohlgfhagcopnndkgoommcihmdmk
[3]https://github.com/first20hours/google-10000-english

REFERENCES

[1] I. E. Akkus, R. Chen, M. Hardt, P. Francis, and J. Gehrke. Non-tracking Web Analytics. In *ACM CCS*, 2012.
[2] F. Bao, R. H. Deng, and H. Zhu. Variations of Diffie-Hellman problem. In *ICICS*, 2003.
[3] N. Borisov, G. Danezis, and I. Goldberg. DP5: A Private Presence Service. In *PoPETS*, 2015.
[4] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient Aggregation of encrypted data in Wireless Sensor Networks. In *Mobiquitous*, 2005.
[5] T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *Financial Cryptography*, 2012.
[6] R. Chen, I. E. Akkus, and P. Francis. SplitX: High-performance Private Analytics. In *SIGCOMM*, 2013.
[7] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards statistical queries over distributed private user data. In *NSDI*, 2012.
[8] H. Choi and H. Varian. Predicting the Present with Google Trends. *Economic Record*, 88(s1), 2012.
[9] :chutten. Two Days, or How Long Until the Data is In. Online at https://blog.mozilla.org/data/2017/09/19/two-days-or-how-long-until-the-data-is-in/, 2017.
[10] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *NSDI*, 2017.
[11] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6), 1976.
[12] C. Dwork. Differential Privacy. In *ICALP*, 2006.
[13] U. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM CCS*, 2014.
[14] M. Ettredge, J. Gerdes, and G. Karuga. Using Web-based Search Data to Predict Macroeconomic Statistics. *Communications of the ACM*, 48(11), 2005.
[15] L. Fan and H. Jin. A Practical Framework for Privacy-Preserving Data Analytics. In *The World Wide Web Conference*, 2015.
[16] G. Fanti, V. Pihur, and ÃŽlfar Erlingsson. Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries. In *PoPETS*, 2016.
[17] A. Gervais, R. Shokri, A. Singla, S. Capkun, and V. Lenders. Quantifying Web-Search Privacy. In *ACM CCS*, 2014.
[18] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232), 2009.
[19] S. Goel, J. M. Hofman, S. Lahaie, D. M. Pennock, and D. J. Watts. Predicting consumer behavior with Web search. *Proceedings of the National Academy of Sciences*, 107(41), 2010.
[20] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly Aggregation for the Smart-grid. In *PETS*, 2011.
[21] V. Lampos, A. C. Miller, S. Crossan, and C. Stefansen. Advances in Nowcasting Influenza-like Illness Rates using Search Query Logs. *Scientific Reports*, 5(12760), 2015.
[22] V. Lampos, E. Yom-Tov, R. Pebody, and I. J. Cox. Assessing the Impact of a Health Intervention via User-Generated Internet Content. *Data Mining and Knowledge Discovery*, 29(5), 2015.
[23] V. Lampos, B. Zou, and I. J. Cox. Enhancing Feature Selection Using Word Embeddings: The Case of Flu Surveillance. In *The World Wide Web Conference*, 2017.
[24] F. J. Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253), 1951.
[25] F. D. McSherry. Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis. In *SIGMOD*, 2009.
[26] L. Melis, G. Danezis, and E. De Cristofaro. Efficient Private Statistics with Succinct Sketches. In *NDSS*, 2016.
[27] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. GUPT: Privacy Preserving Data Analysis Made Easy. In *SIGMOD '12*, 2012.
[28] Mozilla Labs. A Week in the Life of a Browser: Aggregated Data Sample. https://web.archive.org/web/20110711092459/https://testpilot.mozillalabs.com/testcases/a-week-life/aggregated-data.html, 2011.
[29] S. Palan and C. Schitter. Prolific.ac – A subject pool for online experiments. *Journal of Behavioral and Experimental Finance*, 17, 2018.
[30] J. Paparrizos, R. W. White, and E. Horvitz. Screening for Pancreatic Adenocarcinoma Using Signals From Web Search Logs: Feasibility Study and Results. *Journal of Oncology Practice*, 12(8), 2016.
[31] V. Pinchin. I'm Feeling Yucky :(Searching for symptoms on Google. Online at https://www.blog.google/products/search/im-feeling-yucky-searching-for-symptoms/, 2016.
[32] P. M. Polgreen, Y. Chen, D. M. Pennock, F. D. Nelson, and R. A. Weinstein. Using Internet Searches for Influenza Surveillance. *Clin. Infect. Dis.*, 47(11):1443–1448, 2008.
[33] A. Pyrgelis, E. De Cristofaro, and G. J. Ross. Privacy-friendly mobility analytics using aggregate location data. In *SIGSPATIAL*, 2016.
[34] A. Pyrgelis, C. Troncoso, and E. De Cristofaro. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. In *NDSS*, 2018.
[35] Y. Research. L18 - Anonymized Yahoo! Search Logs with Relevance Judgments. Online at https://webscope.sandbox.yahoo.com/catalog.php?datatype=l.
[36] L. Soldaini and E. Yom-Tov. Inferring Individual Attributes from Search Engine Queries and Auxiliary Information. In *The World Wide Web Conference*, 2017.
[37] D. Sullivan. Google now handles at least 2 trillion searches per year. Online at https://searchengineland.com/google-now-handles-2-999-trillion-searches-per-year-250247, 2016.
[38] M. Wagner, V. Lampos, E. Yom-Tov, R. Pebody, and I. J. Cox. Estimating the Population Impact of a New Pediatric Influenza Vaccination Program in England Using Social Media Content. *Journal of Medical Internet Research*, 19(12), 2017.
[39] R. White and E. Horvitz. Evaluation of the feasibility of screening patients for early signs of lung carcinoma in web search logs. *JAMA Oncology*, 3(3), 2017.
[40] L. Wu and E. Brynjolfsson. *The Future of Prediction: How Google Searches Foreshadow Housing Prices and Sales*. University of Chicago Press, 2015.
[41] S. Yang, M. Santillana, and S. C. Kou. Accurate Estimation of Influenza Epidemics using Google Search Data via ARGO. *Proceedings of the National Academy of Sciences*, 112(47), 2015.
[42] E. Yom-Tov. *Crowdsourced Health – How What You Do on the Internet Will Improve Medicine*. MIT Press, 2016.
[43] S. T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys Tutorials*, 15(4), 2013.
[44] T. Zeller. AOL executive quits after posting of search data. https://web.archive.org/web/20061126162350/http://www.iht.com/articles/2006/08/22/business/aol.php, 2006.
[45] P. Zhao and B. Yu. On Model Selection Consistency of Lasso. *Journal of Machine Learning Research*, 7, 2006.
[46] B. Zou, V. Lampos, and I. Cox. Multi-Task Learning Improves Disease Models from Web Search. In *The World Wide Web Conference*, 2018.
[47] H. Zou and T. Hastie. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2), 2005.