Non-tracking Web Analytics

Istemi Ekin Akkus
MPI-SWS
iakkus@mpi-sws.org

Ruichuan Chen
MPI-SWS
rchen@mpi-sws.org

Michaela Hardt
Twitter Inc.
mila@twitter.com

Paul Francis
MPI-SWS
francis@mpi-sws.org

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

ABSTRACT

Today, websites commonly use third party web analytics services to obtain aggregate information about users that visit their sites. This information includes demographics and visits to other sites as well as user behavior within their own sites. Unfortunately, to obtain this aggregate information, web analytics services track individual user browsing behavior across the web. This violation of user privacy has been strongly criticized, resulting in tools that block such tracking as well as anti-tracking legislation and standards such as Do-Not-Track. These efforts, while improving user privacy, degrade the quality of web analytics. This paper presents the first design of a system that provides web analytics without tracking. The system gives users differential privacy guarantees, can provide better quality analytics than current services, requires no new organizational players, and is practical to deploy. This paper describes and analyzes the design, gives performance benchmarks, and presents our implementation and deployment across several hundred users.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection (D.4.6, K.4.2)

Keywords

Tracking, Web Analytics, Differential Privacy

1. INTRODUCTION

Website publishers use web analytics information to analyze their traffic and optimize their site's content accordingly. Publishers can obtain analytics data by running their own web analytics software programs [3, 14, 15]. These analytics programs provide publishers with statistics about users on their site, such as their pageviews, clickstreams, browsers, operating systems, plugins as well as frequency of returning visitors. However, they do not provide other potentially useful information, such as user demographics.

For this reason, publishers often outsource the collection of web analytics to a third party data aggregator, such as comScore, Google, Quantcast or StatCounter [2,24]. A data aggregator collects data from users visiting a publisher's website and presents these data in aggregate form to the publisher. This outsourcing is convenient for publishers, because they only have to install a small piece of code (i.e., a JavaScript code snippet) provided by the data aggregator. More importantly, this scheme allows publishers to learn statistical information they could not otherwise learn from their own web server logs, such as the demographic profile of their user base and the other websites their users visit. A data aggregator can *infer* this *extended web analytics* information because it collects user data across many publisher websites. Compiling extended web analytics via these collected data also benefits the data aggregator because it can sell this information to advertisers and publishers alike.

Although this scheme is beneficial for the publishers and the data aggregators, it raises concerns about users being tracked while browsing the web. This tracking enables a data aggregator to compile detailed behavior of individual users, and infer individual user demographics [48]. Thus, data aggregators are given a lot of information about users' actions on the web and have to be trusted that they will not abuse it. This trust has been violated in the past [13,17,18].

Tracking affects not only users, but also the data aggregators themselves, who are often criticized for this behavior. These criticisms have led to industry self-regulation to provide opt-out mechanisms [4, 5, 19, 25], the Do-Not-Track (DNT) initiative in the W3C, and many client-side tools, either to implement DNT [10, 20, 21], or to prevent tracking outright [1, 7, 9, 11]. To the extent that these efforts take hold, the ability for data aggregators to provide extended analytics to publishers will be degraded.

In addition, even with tracking, inferring accurate user demographics is a difficult task that may produce inconsistent results. For example, according to Quantcast, 24% of rottentomatoes.com's visitors in US are between 18 and 24, and 20% are between 35 and 44 [23], whereas Doubleclick says these numbers are 10% and 36%, respectively [6].

To address these issues, we present the design and implementation of a practical, private and non-tracking web analytics system. Our system allows publishers to *directly measure* extended web analytics rather than rely on inferred data, while providing users with differential privacy guarantees under a set of realistic threat assumptions. These

CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.

guarantees enable aggregation of users' private information, such as demographics and websites visited, without violating individual user privacy.

In our system, user information is stored in a database on the user device (client). We exploit the direct communication that naturally takes place between the publisher and the users by having the publisher distribute database queries to clients, and by having the publisher act as an anonymizing proxy for the (encrypted) answers from the clients to the data aggregator. The aggregator aggregates the anonymous answers, and provides the results to the publisher. Both the publisher and the aggregator add differentially-private (DP) noise before passing data on to each other.

The decision to use the publisher as a proxy allows us to avoid introducing a new organizational player in the architecture. While this decision is good for deployability, it creates new technical challenges because publishers can be malicious. In particular, they might try to exploit their position in the middle by manipulating which clients receive and answer queries, and to overcome the DP noise added by the aggregator using repeated queries.

The contributions of this paper are as follows. To the best of our knowledge, we are the first to study the problem of collecting extended web analytics information without tracking users. We describe and analyze a novel design that provides the first practical solution to this problem, and prove that our design provides differential privacy. Finally, we implement and evaluate our system to gauge its feasibility.

The next section presents our goals and trust assumptions. After giving an overview of our system in Section 3, we describe its details in Section 4. We analyze our system's properties in Section 5, and report our implementation and evaluation in Section 6. Section 7 describes related work. We discuss future work and conclude in Section 8.

2. GOALS AND ASSUMPTIONS

2.1 Functionality Goals

There are three entities in today's tracking web analytics systems: the publisher, the data aggregator, and the client. Publishers create websites. Data aggregators provide publishers with aggregation service for web analytics. Users use their clients (e.g., the browser) to access and consume the content that publishers host.

Figure 1 shows the interactions between these entities today. When clients visit the publisher's website (step 0), they also send analytics data to the data aggregator via the code snippet installed on the publisher's website (step 1). After collecting information from individual clients, the data aggregator aggregates analytics information (step 2), and then shares the aggregate result with the publisher (step 3).

Broadly speaking, and putting user privacy aside for the moment, we would like our system to provide publishers and data aggregators with *at least* the benefits they enjoy in today's systems. We would also like to avoid requiring new players like proxies. Specifically, our functionality goals are:

1. Publishers should get more accurate and more types of web analytics information than they do today.

2. Data aggregators should obtain web analytics information for all of their partner publishers like they do today, as an incentive for performing aggregation.

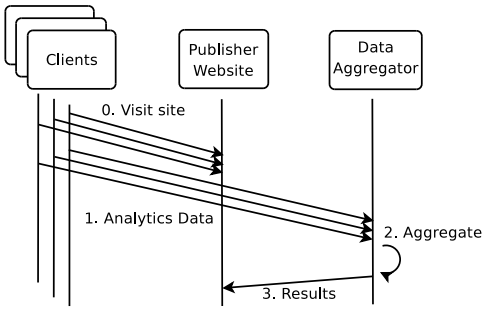

Figure 1: Operation of today's tracking web analytics systems.

3. The system should scale adequately, and ideally be as or more efficient than today's systems.

4. The system should not allow clients or publishers to manipulate results beyond what is possible today (i.e., via botnets).

2.2 User Privacy Goals

Our main user privacy goal is that user data should be protected within the formal guarantees of differential privacy (DP) [36]. In particular, each client should know its own privacy loss as defined by DP with respect to each publisher and each data aggregator. While our system provides this guarantee, it should be noted that such knowledge at the client is of limited value. DP is very conservative, because it assumes that the attacker may have arbitrary auxiliary information it can use to discover information about users in the database. When the attacker does not have this auxiliary information, which is the common case, DP's measure of privacy loss is overly pessimistic. Although a client in our system could in theory refuse to answer queries if a privacy budget is exceeded, doing so is not practical in our setting, because a query may be legitimately repeated from time to time (e.g., to measure changes in the user base). Furthermore, DP's privacy loss measure assumes a static database, whereas in our setting, the "database" is dynamic: the user population for a given publisher changes almost constantly, and some individual user data may change as time passes.

For these reasons, it is unrealistic, and in our setting, unnecessary to set a hard limit on user privacy loss (i.e., budget). Nevertheless, we find DP to be a valuable mechanism, in part because it provides a worst-case measure of privacy loss, but primarily because the noise added to answers substantially raises the bar for the attacker, while still providing adequate accuracy for aggregate results.

2.3 Non-goals

Publishers today can directly measure user activity on their websites (e.g., pages visited, links clicked). In addition, websites can often legitimately obtain additional information directly from users, such as personally identifiable information (PII), shopping activity, friends, and hobbies. Information obtained directly from users by publishers is considered outside the scope for the purposes of this paper.

Note that today most data aggregators provide behavioral advertising, for which they require individual user information, and not aggregate data. Given that other research

shows how to accomplish behavioral advertising without exposing user information [40, 51], for this paper we assume that the data aggregator requires only aggregate data.

2.4 Trust Assumptions

2.4.1 Client

We assume that the users trust the client software, in terms of the data it stores locally and its operation, just as they trust their browser software today. While it is possible for a browser to be infected by malware, such malware is in a position to violate user privacy in many ways beyond our system; thus, we do not protect against this threat.

By contrast, we assume that the clients may be malicious towards the publisher and the data aggregator. A malicious client may attempt to distort aggregate results, similar to the situation today where a client may, for instance, participate in click fraud. It may also try to violate the privacy of other users, possibly colluding with the publisher.

2.4.2 Data Aggregator

We assume that the data aggregator is honest-but-curious (HbC); in other words, that it obeys the prescribed operation, but may try to exploit any information learned in the process. As an HbC player, we assume that the data aggregator does not collude with the publishers. In principle, a malicious publisher could of course simply choose to work with a malicious data aggregator. We assume a setup whereby aggregators state their non-collusion in a privacy statement, making them legally liable and subject to punishment (e.g., by the FTC). An aggregator that is also a publisher would have to internally separate information. We justify an HbC aggregator on the assumption that the client software plays an overseer role, and allows only HbC aggregators to participate. For instance, the browser could refuse inclusion to any aggregator that does not provide such a privacy statement, or appears untrustworthy for any other reason. Today, browsers already play a similar role in a number of respects, such as by selecting default certificate authorities and search engines, and in some cases, by warning users of potentially harmful websites. In today's industry setting where major data aggregators can generally be expected to operate within the legal framework of their own stated privacy policies, we think that this assumption is reasonable.

2.4.3 Publisher

We assume that the publisher is selfishly malicious both towards the users and the data aggregator, meaning that the publisher will try to only benefit itself. As a potentially malicious player, the publisher may try to violate the privacy of users with correct clients. In particular, because the publisher distributes queries and collects answers, it is in a position to selectively query clients, drop selected client answers, and add answers beyond those required for DP noise. This position leads, for instance, to an attack whereby the publisher *isolates* a single client by dropping all answers except for those of the single client, and providing fake answers instead of the dropped answers. With repeated queries to such an isolated client, the publisher may overcome the added DP noise. The publisher may also be motivated to falsify the results it gives to the data aggregator, for instance, to appear more popular or more attractive to advertisers. Our design has mechanisms to mitigate the effect of these behaviors.

Note, however, that we assume that the publisher correctly adds DP noise to answers, because withholding noise does not benefit the publisher, and the minor reduction in overhead gained (Section 6) is not adequate incentive.

2.5 Incentives

The incentives for the publisher and the aggregator are richer (Section 6), and more accurate analytics, because we directly measure attributes rather than infer them. We do not think that users are incentivized. Although publishers could offer incentives to users (e.g., better content for participating users) to create an incremental deployment environment, we think that the browser is a better option for deployment. The publisher, aggregator, and browser should also be motivated to provide better privacy to users. Even though we do not know for certain whether our stated incentives are adequate, we think that they are at least feasible.

3. SYSTEM OVERVIEW

Our system comprises the same three entities that exist today: the client, the publisher, and the data aggregator (Figure 2). The publisher plays an expanded role: it distributes queries to clients, and it proxies client-aggregator communication. This role requires that the publisher, or its hosting center, to install new software. While this requirement reduces ease-of-use compared to today, we think it is reasonable: many publishers already run their own analytics software [2, 3, 14, 15, 24] and hosting companies already offer servers with web analytics software pre-installed [8, 12, 16].

The client gathers and stores information in a local database, and answers publishers' queries using this database. This information consists of, for instance, user demographics, browsing behavior, or non-user related information, like system performance. Demographics information can include age, gender, marital status, education level, location and income. Browsing behavior can include pages visited, purchases, and searches made. We envision that the client scrapes most of this information from web pages the user visits (with informed user consent), such as online social networks, shopping websites, and search engines. The user can also provide some information directly, or the client can ultimately infer some information, like income. This scraping functionality can be supported by the browser. For instance, the browser may implement basic messaging, encryption and database mechanisms, and provide a sandboxed plugin environment for aggregators' clients.

To distribute queries to clients, publishers post queries at well-known URLs on their websites. When clients visit a website (step 0 in Figure 2), they read the queries (step 1).

Queries may be formulated by both the publisher and the data aggregator. While the queries themselves may be quite complex (i.e., SQL), the answers are limited to 'yes' and 'no' values. For instance, for the age distribution of users, the query effectively asks clients to evaluate 'yes' or 'no' for each age range of interest (e.g., <18, 18-34, 35-50, >50). This answering mechanism is achieved by defining *buckets*, such that each bucket corresponds to a potential answer value, and by mapping the query result to these buckets. Ultimately, the aggregator generates a per-bucket histogram of user counts. The primary benefit of using buckets is to limit the distortion a malicious client can impose on the aggregate result.

Each generated answer is separately encrypted with the public key of the data aggregator (step 2). Queries may

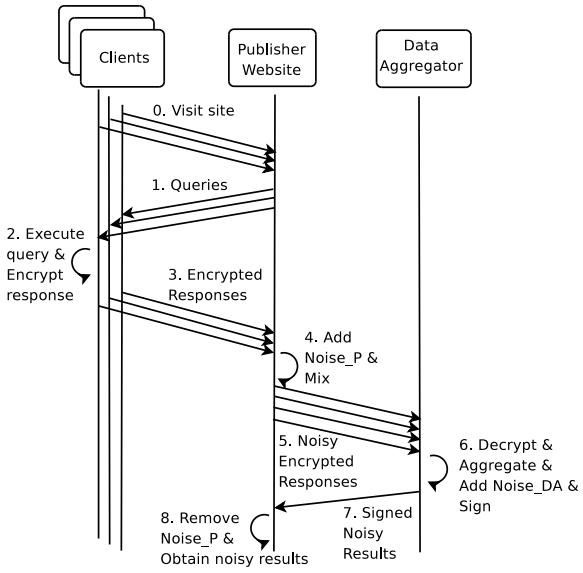

Figure 2: Query workflow of our system.

have thousands of defined buckets, most of which have 'no' answers; for instance, one for each website a user may visit, or for each interest a user may have. To reduce the number of crypto operations, 'no' answers are omitted at the client. Instead, clients generate a specified number of answers which are either 'yes' or 'null'. For example, a query may specify that every client produces exactly 20 answers for the websites a user has visited in the last week, regardless of the actual number of visited websites. If a client has not visited 20 different websites, it generates 'null' answers for the difference. If it has visited more than 20 websites, then it cannot report on every website visited.

After collecting the encrypted answers from clients (step 3), the publisher generates DP noise separately for each bucket in the form of additional answers. It then mixes the real and the noise answers (step 4) and forwards all answers to the data aggregator (step 5).

The data aggregator decrypts the answers, computes the histogram of bucket counts, and adds DP noise to each count (step 6). After signing the result, it transmits the counts to the publisher (step 7), who finally subtracts the noise it originally added to obtain its own final counts (step 8).

In the end, the publisher and the data aggregator *both* obtain aggregate results for the query. Because of the noise, neither of them obtains an exact result: the publisher's result contains the noise the aggregator added, whereas the aggregator's result contains the noise the publisher added.

If the publisher or the data aggregator wishes to release a result to the public, then they must release the "double noisy" result that was passed to the publisher in step 7. This precaution prevents the publisher and the aggregator from computing the noise-free result by subtracting their own noise, should the other publish its "single noisy" result.

3.1 Audits

Clients occasionally *audit* publishers to detect if a publisher is dropping client answers (Figure 3). To audit a publisher, the client generates and encrypts a nonce (Step 2),

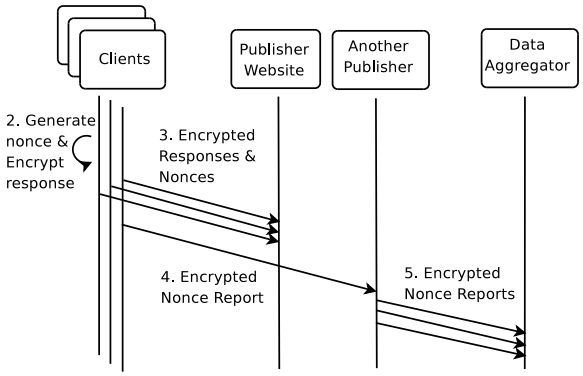

Figure 3: Auditing mechanism of our system.

and transmits it to the publisher instead of the answer the client otherwise would have sent (Step 3). The client also transmits the nonce to another, randomly selected publisher (Step 4), which forwards this *nonce report* to the data aggregator (Step 5). If the data aggregator often receives nonce reports without the corresponding nonce answer, it suspects the publisher of dropping client answers.

4. DESIGN DETAILS

In this section, we describe how queries are generated and distributed, how the client generates a response and helps in auditing publishers, and how DP noise is added by the publisher and the data aggregator.

4.1 Differential Privacy Background

A computation, $\mathcal{C}$, provides (ϵ, δ)-differential privacy [36] if it satisfies the following inequality for all datasets D_1 and D_2 differing on one record and for all outputs $S \subseteq Range(\mathcal{C})$:

$$\Pr[\mathcal{C}(D_1) \in S] \leq \exp(\epsilon) \times \Pr[\mathcal{C}(D_2) \in S] + \delta \qquad (1)$$

In other words, the probability that a computation $\mathcal{C}$ produces a given output is almost independent of the existence of any individual record in the dataset. In our setting, this dataset consists of the values of clients for a given attribute.

Differential privacy is achieved by adding noise to the output of the computation. This noise is independently generated for each component in the dataset. There are two privacy parameters: ϵ and δ. The trade-off between the accuracy of a computation and the strength of its privacy guarantee is mainly controlled by ϵ: a smaller ϵ provides higher privacy, but lower accuracy.

The parameter δ relaxes the strict relative shift of probability. If $\delta = 0$, then the (ϵ, δ)-differential privacy falls back to the classical ϵ-differential privacy, which can be achieved by adding the Laplace distribution noise with a standard deviation $\sqrt{2}\Delta\mathcal{C}/\epsilon$, where $\Delta\mathcal{C}$ is the sensitivity of the computation, and is 1 for a computation counting set elements [35].

A non-zero δ is required in some scenarios where the inequality (1) cannot be satisfied [36]. Such (ϵ, δ)-differential privacy can be achieved in our system by adding the aforementioned Laplace distribution noise with a complementary resampling mechanism (Section 4.5.1, and Appendix A).

4.2 Queries

Publishers are required to list all their queries at a well-known URL on their website. This query list is signed by the data aggregator, even if there are no queries (i.e., an empty list). The aggregator may periodically check to ensure that the list is posted at the well-known URL (e.g., via fake clients it has deployed) to detect malicious publishers isolating clients by controlling the distribution of queries to clients. When a client visits a website, it retrieves the query list if the previous list has expired. Each query in the list contains the following fields:

QId	Query ID
p_s	Query selection probability
p_a	Audit probability
T_e	Query end time
$\mathcal{B}$	Set of answer values (buckets) each with ID b_i. ('null' and 'Not Applicable' (N/A) are well-known IDs)
A	Required number of answers
ϵ_P	DP noise parameter for the publisher's result (used by the data aggregator)
ϵ_{DA}, δ	DP noise parameters for the data aggregator's result (used by the publisher)
SQL	Database query

QId is unique among all queries across all publishers working with this data aggregator. For each query in the list, the client decides whether to answer or ignore the query. This decision is made with the *selection probability* p_s assigned by the publisher, such that the publisher can obtain enough answers from its user base given the expected number of client visits. δ could be computed based on the expected number of answers. If the client decides to answer the query, then it separately decides whether to audit the query with *audit probability* p_a assigned by the data aggregator, by replacing the answer with a nonce. The *query end time* is the deadline for answering queries.

The buckets $\mathcal{B}$ may be pre-defined, downloaded separately, and cached, if $|\mathcal{B}|$ is large. The aggregator sets the DP noise parameter ϵ_{DA}, and checks each query's ϵ_P to ensure it generates adequate noise before signing the query list.

The SQL query may produce zero or more numerical values or strings. Each bucket is defined as a numerical range or a string regular expression, such as salary ranges (numerical), or websites visited (string). A bucket is labeled as 'yes' if a row in the SQL output falls within the numerical range, or matches the regular expression. In addition, buckets have instructions to be followed when the same SQL output labels multiple buckets as 'yes' (e.g., select one or all), and the number of 'yes' labeled buckets exceeds the allowed number of answers A (e.g., select most frequently occurring or random buckets). If the client does not have A number of 'yes' labeled buckets, it uses a well-known bucket ID 'null'.

As an example, suppose a publisher wants to learn the age distribution of its female users. The SQL can be "`SELECT age FROM LOCAL_DB WHERE gender = female`". The buckets can be $\mathcal{B} = \{< 18, 18-34, 35-50, > 50\}$, and $A = 1$.

An SQL query often has predicates, such as "`WHERE gender = female`" in the above example. These predicates enable the publisher to query different segments of its user base. Too specific predicates, however, may produce results that are not useful in aggregate. To enable the publisher to notice that the predicates are too narrow, we define another well-known bucket ID 'Not Applicable' ('N/A'), used by the client when the query predicates fail. This well-known bucket ID is also useful for the data aggregator to detect malicious publishers who may set very specific predicates to isolate a client and repeat the query to overcome the noise (e.g., few answers with bucket IDs other than 'N/A').

4.3 Query Response

If a client decides to answer a query, it executes the SQL on its local database (step 2 in Figure 2) and produces the set of buckets labeled 'yes' (i.e., $\mathcal{M}$). If no predicates match, the client generates A answers with the well-known bucket ID 'N/A', each individually encrypted with the public key of the data aggregator:

$$Response = Enc_{DA\_pub}\{QId, N/A\} \quad (A \quad times)$$

If the predicates match and $|\mathcal{M}| \leq A$, the client produces A individually encrypted answers, where $|\mathcal{M}|$ answers contain the matching bucket ID $b_i \in \mathcal{M}$, and $A - |\mathcal{M}|$ answers contain the well-known bucket ID 'null' :

$$Response = \begin{cases} Enc_{DA\_pub}\{QId, b_i\} & \forall b_i \in \mathcal{M} \\ Enc_{DA\_pub}\{QId, null\} & (A - |\mathcal{M}| \, times) \end{cases}$$

If $|\mathcal{M}| > A$, the client selects A buckets according to the instructions and produces A individually encrypted answers.

After generating the response, the client transmits it to the publisher along with the query ID (step 3 in Figure 2):

$$C \rightarrow P : QId, Response$$

To illustrate, assume a query asks the 20 most visited sites. If a client visited only 14 sites, it generates 14 answers with bucket IDs representing these sites and six answers with bucket ID 'null'. In contrast, if the client visited 25 sites, it generates only 20 answers for the 20 most visited sites.

The client then records that it answered the query so as not to answer it again before the query end time T_e. It also records the ϵ values to track the user's privacy exposure to the publisher and the aggregator. Note that the publisher may store client IP addresses answering this query to prevent a malicious client from skewing the aggregate result by sending many responses for the same query.

4.4 Audit Response

The audit serves two purposes. First, it can detect when a publisher is dropping client answers. Second, it can detect when a publisher is adding a substantial number of fake answers (beyond the noise).

If a client decides to audit the publisher, it generates a nonce, encrypts the nonce and the QId with the aggregator's public key as well as $A-1$ 'null' answers, and transmits the response to the publisher as if it was a real response:

$$Response = \begin{cases} Enc_{DA\_pub}\{QId, nonce\} & once \\ Enc_{DA\_pub}\{QId, null\} & (A-1 \quad times) \end{cases}$$

The client also obtains a blind signature *blind_sig* for the nonce from the aggregator [31] and randomly selects a different publisher, which is a customer of the aggregator. The client then transmits a separate, encrypted copy of the nonce and the nonce's blind signature to that publisher:

$$NR = (Enc_{DA}\{QId, nonce\}, blind\_sig)$$

This nonce report cannot be directly submitted to the data aggregator, because the aggregator would learn which publisher a client has visited and decided to audit.

Each publisher periodically forwards received nonce reports to the aggregator. The client learns the set of other publishers by periodically downloading a list from the aggregator. This list associates a probability with each publisher that is roughly proportional to the number of answers each publisher handles. The client selects the different publisher according to this probability. As a result, each publisher handles a fair proportion of nonce reports.

If the aggregator consistently receives nonce reports via different publishers without a corresponding nonce message from the audited publisher, the aggregator suspects the audited publisher of dropping messages, possibly in an attempt to isolate a client. In this case, the aggregator can validate this suspicion by masquerading as real clients from browsers it controls, and sending audits from these clients. This check is necessary because a malicious client may have sent nonce reports via different publishers, without the corresponding nonce via the audited publisher to cast suspicion on it.

Because the aggregator knows the probability of sending an audit response instead of a query response, it knows the proportion of audits to answers that should be received. If this proportion is consistently too low, then the aggregator suspects the publisher of adding additional fake answers.

The purpose of the blind signatures is to limit the rate a client can generate audits, which is helpful for two reasons. First, it drastically reduces the amount of suspicion a malicious client can cast on publishers by just sending the nonce reports, but not the nonces, as described above.

Second, by ensuring that these blind signatures are only assigned to clients and not publishers, the data aggregator prevents a malicious publisher from trivially generating many fake audits. Using these fake audits, the publisher could either drop client answers for an isolation attack without considering the possibility that they may be audit responses, or generate many fake answers by maintaining the right proportion of audits to answers. This use of blind signatures ultimately raises the bar for the publisher by forcing it to use botnet clients.

The blind signatures are timestamped to prevent an attacker from hoarding them for later use [26, 30, 38]. These timestamps are coarse-grained (e.g., end of the week) to prevent the aggregator from linking signatures to clients.

4.5 Noise Generation

4.5.1 Noise at the Publisher

The publisher generates differentially-private (DP) noise, rounded to the nearest integer, for *all* buckets with the data aggregator's ϵ value (i.e., ϵ_{DA}), $\mathcal{N}_P = \{n_{P,1}, n_{P,2}, ..., n_{P,b}\}$, where b is the number of buckets (shown as Noise_P in step 4 in Figure 2). The mechanism for generating noise is to create additional answers. The amount of noise to add may of course be positive or negative. Recall, however, that the answers themselves are positive bucket counts. There is no way to supply a negative bucket count. To generate negative noise, we define an offset value o, which the aggregator will *subtract* from each per-bucket count. The number of additional answers supplied will be greater or less than this offset to create positive or negative noise, respectively.

To give an example, if the offset is 20, and the noise is +4, the publisher creates 24 answers for the given bucket, and the aggregator later subtracts 20 from the bucket's count. On the other hand, if the noise is -5, the publisher creates 15 answers. Stated precisely, the publisher calculates the number of per-bucket answers to create as:

$$\begin{aligned}\mathcal{N}_{\mathcal{P}}^{'} &= \{n_{P,1}+o, n_{P,2}+o, ..., n_{P,b}+o\} \\ &= \{n_{P,1}^{'}, n_{P,2}^{'}, ..., n_{P,b}^{'}\}\end{aligned}$$

These noise answers are encrypted with the aggregator's public key; hence, indistinguishable from client answers. After the query end time T_e, the combined set of client answers and noise answers $\mathcal{R}_{DA}$ are randomly mixed and sent to the aggregator along with the query ID and offset value:

$$P \rightarrow DA : QId, \mathcal{R}_{DA}, o$$

The aggregator decrypts the answers, counts them, and subtracts the offset to obtain the noisy result:

$$\begin{aligned}\mathcal{R}^{'}{}_{DA} &= \{r_1 + n_{P,1}^{'} - o, r_2 + n_{P,2}^{'} - o, \\ &\quad ..., r_b + n_{P,b}^{'} - o\} \\ &= \{r_1 + n_{P,1}, r_2 + n_{P,2}, ..., r_b + n_{P,b}\}\end{aligned}$$

where r_i is the count of client answers belonging to bucket b_i, and $n_{P,i}$ is the publisher's noise value for bucket b_i.

At this point, the data aggregator can make two checks to detect potential malicious publisher behavior. First, the aggregator can estimate the number of expected answers based on the number of audits received for this query and the audit probability p_a. After accounting for the expected noise answers (i.e., $b \times o$), if the received number of answers is significantly higher or lower than the expected number of answers, the aggregator suspects the publisher of adding or removing answers, respectively.

Second, after obtaining the bucket counts, the data aggregator can check for anomalies in this publisher's results. For instance, if the results for the same query consistently show low-value buckets along with high-value buckets (e.g., 'female<3', 'N/A>1K'), the publisher may be trying to isolate a client's answer and overcome the noise. In this case, the aggregator may suspect the publisher, check the query predicates manually, and/or may not return the result.

There remains the question of how to set the value of o. The noise value cannot exceed the offset, and must be resampled when $n_{P,i} < -o$. In Appendix A, we prove that even with this resampling, our procedure still provides (ϵ, δ)-differential privacy. Theorem 1 from Appendix A sets o as:

$$o \geq \lambda \ln\left(\left(e^{\frac{A}{\lambda}} - 1 + \delta/(2A)\right) A/\delta\right) \quad (2)$$

where $\lambda \geq 2A/\epsilon_{DA}$. [43] argues that, for DP guarantees to be met, $\delta < 1/c$, where c represents the number of clients answering this query. Since in our setting a client may answer the same query multiple times, we require $\delta < 1/(m \times c)$, where m represents the maximum number of queries a client can answer. For the purpose of this paper, we assume a conservative setting of $m = 1000$.

4.5.2 Noise at the Data Aggregator

After aggregating the answers and obtaining the noisy results, $\mathcal{R}^{'}{}_{DA}$, the data aggregator generates DP noise using the ϵ value specified by the publisher (i.e., ϵ_P), $\mathcal{N}_{DA} = \{n_{DA,1}, n_{DA,2}, ..., n_{DA,b}\}$ (shown as Noise_DA in step 6 in

Figure 2). The data aggregator computes the results $\mathcal{R}_P$:

$$\begin{aligned}\mathcal{R}_P &= \mathcal{R}^{'}{}_{DA} + \{n_{DA,1}, n_{DA,2}, ..., n_{DA,b}\} \\ &= \{r_1 + n_{P,1} + n_{DA,1}, r_2 + n_{P,2} + n_{DA,2}, \\ &\quad ..., r_b + n_{P,b} + n_{DA,b}\}\end{aligned}$$

Then, this result is signed by the data aggregator and sent to the publisher (step 7 in Figure 2):

$$DA \rightarrow P : QId, \mathcal{R}_P$$

When the publisher gets $\mathcal{R}_P$, it removes its own noise and obtains its own noisy results, $\mathcal{R}^{'}{}_P$, (step 8 in Figure 2):

$$\begin{aligned}\mathcal{R}^{'}{}_P &= \mathcal{R}_P - \{n_{P,1}, n_{P,2}, ..., n_{P,b}\} \\ &= \{r_1 + n_{DA,1}, r_2 + n_{DA,2}, ..., r_b + n_{DA,b}\}\end{aligned}$$

where r_i is the count of client answers belonging to bucket b_i, and $n_{DA,i}$ is the aggregator's noise value for bucket b_i. In the end, the aggregator's result contains the DP noise added by the publisher (i.e., $R^{'}_{DA}$), whereas the publisher's result contains the DP noise added by the aggregator (i.e., $R^{'}_{P}$).

5. ANALYSIS

5.1 Data Aggregator

Although the data aggregator follows the prescribed operation and does not collude with publishers, it may still be motivated to track clients across publishers and may try to exploit any information it learns. This information can include *identifiers* associated with clients, allowing the aggregator to track them. In the absence of a proxy, one such identifier is the client *IP address*. By using the publisher as an anonymizing proxy, our system hides IP addresses from the aggregator during the collection of answers.

The aggregator may try to obtain other identifiers by manipulating query parameters (i.e., $\epsilon_{DA}, \epsilon_P$, and p_a), and the audit activities (i.e., assignment of blind signatures and publisher probabilities for nonce reports). For example, an answer to a common query (e.g., a rare occupation) can be distinguishing among clients. The DP noise added by the publisher solves this problem (Section 4.5.1). To minimize this noise, the aggregator may set a large ϵ_{DA} value, but it would be easily detected by clients and industry regulators.

Besides rare answers, the *combination of answers* can also act as an identifier for a client. For example, a client's response to a query about most visited sites may be unique. Our system solves this problem by separately encrypting each answer at the client (Section 4.3) and mixing client answers with noise answers at the publisher (Section 4.5.1).

The aggregator has no incentive to use a large ϵ_P, which would only serve to reduce noise for the publisher.

In the auditing mechanism, the nature of the blind signatures and coarse-grained timestamps prevents the aggregator from connecting nonce reports back to clients. The aggregator may set a large audit probability for one publisher, and small probabilities for other publishers. This high probability would cause the clients of the first publisher to obtain blind signatures more often than others; hence, enabling the aggregator to infer the publisher these clients visit. However, unusually high audit probabilities will raise suspicion among clients, and regulators. Furthermore, the utility of the first publisher will suffer, triggering detection.

5.2 Publisher

A potentially malicious publisher may want to exploit its position in the middle to learn an individual client's information, and falsify the results the data aggregator gets. The publisher can control the query parameters (i.e., SQL, A, $\mathcal{B}$, ϵ_P, p_s), the distribution of queries, the collection and forwarding of responses and nonce reports, the noise process, and the publishing of final results. We analyze how a publisher can try to exploit these parameters, and discuss how our system raises the bar for these attempts to succeed.

5.2.1 Publisher Attacking Clients

A client's response is encrypted with the aggregator's public key. The client also sends a fixed number of answers (i.e., A), preventing the publisher from learning how many buckets were matched for a query. Absent collusion, the publisher cannot learn an individual client's answer from the aggregator, and obtains only noisy aggregate results.

To minimize the noise the aggregator adds, the publisher may set a large ϵ_P value. By enforcing a maximum ϵ_P value, the aggregator can ensure that it will add enough noise to protect users' privacy (Section 4.5.2). Nevertheless, a publisher may try to learn a client's answer, by isolating it and repeating the same query to overcome the noise. We discuss how our system raises the bar for such a publisher.

Isolation via selectively dropping other clients' answers: To isolate a client's answer, a malicious publisher may drop answers from other clients and replace them with fake answers it generates. This attempt will be detected by the aggregator, because dropped answers will contain nonces. When the aggregator consistently receives reports via other publishers, but not the nonces from the audited publisher, it suspects the publisher of dropping answers and can confirm this suspicion by masquerading as real clients and sending nonces through the publisher (Section 4.4).

To allow the malicious publisher to drop answers, other colluding publishers may drop nonce reports. However, they cannot selectively do so to help their partners, because they do not know about which publisher a given nonce report is. The aggregator also knows approximately how many reports a publisher should forward (i.e., via the publisher's probability to be randomly selected), and if it does not receive enough reports, it suspects the publisher of dropping them. For these reasons, a publisher cannot easily help another malicious publisher to drop answers without detection.

Isolation via dropping target client's answer: A difficult, but theoretically possible attack is for the publisher to repeat a query and obtain results, half of which contain the target client's answer, and half of which do not. By comparing the average result of these two sets of queries, the publisher can determine if the target client's answer is positive or 'null'. The auditing mechanism may not detect this attack, because the audit is relatively rare, and thus, the target client may generate zero or very few audits. This attack is hard to carry out, because the client population may change over time, and because if the selection probabilty p_s is less than 1, different clients will answer different queries. In both of these cases, the non-noisy value would change a bit with successive queries; thus, requiring even more queries to eliminate the effect of noise.

Nevertheless, we simulated this attack, assuming a fixed set of 100 clients, one of whom is the target client. We execute the same query Q times, varying Q from 30 to 2000.

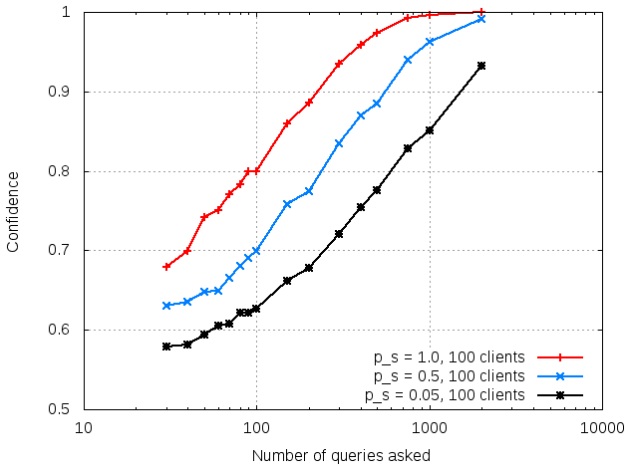

Figure 4: Confidence level for the isolation attack via dropping target's answer.

We also vary the selection probability p_s to be 1.0, 0.5, and 0.05. When $p_s = 1.0$, we drop the target's answer half the time. For $p_s = 0.5$ and $p_s = 0.05$, there is no need to intentionally drop the target answer, because it is often naturally not provided. We average the counts for queries with and without the target. If the difference in the average is greater than 0.5, we guess that the target's answer is 1. If it is less than 0.5, we guess that the target's answer is 'null'. We vary the number of queries Q, run 10000 trials for each Q, and calculate the percentage of times the guess is correct. This percentage is the publisher's confidence after Q queries.

Figure 4 shows the results of our simulation for the cases where the selection probability p_s is 1.0, 0.5, and 0.05. When $p_s = 1.0$, it takes over 350 queries to reach 95% confidence. Assuming one query per week, this attack would take roughly seven years. In the cases of $p_s = 0.5$ and $p_s = 0.05$, the attacker requires about 1000 and 2000 queries, respectively, for the same level of confidence.

Isolation via buckets or SQL: The publisher can also isolate a client by manipulating the query such that only the target client provides a positive answer (or, conversely, all clients except the target client provide positive answers). This attack can be accomplished either by manipulating the SQL predicate, or the bucket definitions (i.e., to include PII or a rare combination of attributes). Our general approach to both of these methods is to monitor answers for clues signaling that this attack may be happening, and to manually inspect SQL queries when these clues appear. While manual inspection is not ideal, given that most publishers will ask the same types of queries, most of the queries will come from an already approved library.

The clue we are searching is any bucket whose count is, for the same repeated query, consistently very low (roughly 0), or very high (roughly the number of answering clients). For instance, if the predicate isolates the user (i.e., the user's name), then we expect to see very low bucket counts, except for the 'N/A' bucket, whose value will be very high. If the predicate does the reverse (i.e., includes all clients, but the target), then the 'N/A' bucket will be very low, and the other buckets will be very high. A very low 'N/A' bucket is suspicious, because in this case the predicate is apparently not needed and should be dropped. Likewise, if the target user is isolated by a rare bucket definition, then certain buckets will have very low counts. In this case, we can expect an honest publisher to modify its bucket definitions to prevent such consistent very low counts.

In some cases, however, examining the SQL may not be adequate. One such example is a predicate like "`WHERE page-visited = example.com/UniqueURL`", where `UniqueURL` is provided only to the isolated client. In this case, the aggregator must check that the URL is provided to multiple clients by operating fake clients.

Isolation via query distribution: A malicious publisher may send a query to only one client. The aggregator, however, can ensure that the queries are available at well-known URLs at the publisher site via its own fake clients. Our auditing mechanism can also be extended to send reports when the query list or any queries are not accessible by clients.

Other attacks: By enforcing a maximum A value, the aggregator can ensure that clients do not spend unnecessary resources (e.g., CPU, bandwidth) while answering queries, to prevent denial of service attacks by the publisher.

There is no clear incentive for a selfishly malicious publisher not to add DP noise. Even so, the aggregator can still detect suspicious behavior: The aggregator can estimate the number of clients c_e as the number of audit reports received divided by the audit probability p_a. The approximate total number of expected answers is therefore $(c_e \times A) + (b \times o)$, where b is the number of buckets, o is the offset, and A is the number of answers per client. If answers are substantially lower than this, the publisher is suspected.

A malicious publisher can publish its own single-noisy results that include only the aggregator's noise. However, these results will not have the aggregator's signature; thus, exposing the publisher. Furthermore, the aggregator can detect this behavior because it knows the double-noisy results.

5.2.2 *Publisher Falsifying Results*

To appear more popular or more attractive to advertisers, a publisher may want to falsify results by generating many fake answers. If the publisher exceeds the number of answers expected by the aggregator (i.e., $c_e \times A + b \times o$), it will be suspected. Thus, it can only generate answers that belong to certain buckets and is limited by the number of buckets and the offset (i.e., $b \times o$). This number may not be significant for queries with few buckets, depending on the total number of answers. For instance, it is 100 for a query about gender distribution with 5000 answers and an offset of 50.

On the other hand, $b \times o$ can be large for queries with many buckets. If all fake answers are used in few buckets, all other buckets would have values close to $-o$ after the offset subtraction. The probability of simultaneously generating these noise values is extremely low, signaling a manipulation. To prevent detection, the publisher would distribute the fake answers more evenly, limiting its distortion in a bucket.

5.3 Client

A client may act maliciously towards the publisher, the aggregator and to other honest clients. A client can lie in its response to distort the aggregate result; however, this distortion is limited by A set by the publisher. By keeping a record of client IP addresses, the publisher can also ensure that a client sends only one response for a query.

Table 1: Queries and associated parameters. The buckets include our two well-known bucket IDs.

Property	# buckets	A	o
Age	7 + 2	1	66
Gender	2 + 2	1	66
Income	6 + 2	1	66
Education	5 + 2	1	66
Has children?	2 + 2	1	66
Location	5000 + 2	1	66
Ethnicity	5 + 2	1	66
Other sites visited	3000 + 2	10	751
Total # pages visited	10 + 2	1	66
Visit frequency	(1000 × 3) + 2	10	751
Search engines used	5 + 2	3	211

By sending fake nonce reports without the corresponding nonces, a malicious client can incriminate a publisher, and cause the aggregator to manually check this publisher. A client may also collude with a malicious publisher and generate nonce reports to help the publisher maintain the right proportion of audits to answers, either in an isolation attack, or in generation of fake answers to falsify results. By controlling the blind signature assignment to clients, the aggregator can limit this behavior, and force the publisher to get a bigger botnet, increasing chances of detection.

6. IMPLEMENTATION & EVALUATION

6.1 Implementation

We implemented the **client** as a Firefox addon. Our client keeps user information in a local database, looks for queries at a well-known URL (e.g., publishersite.com/queries/), and returns an encrypted response. The client is about 1000 lines of JavaScript code and 3000 lines of RSA libraries.

The **publisher software** consists of a simple server-side script that stores the encrypted responses at the publisher's website, and a plugin for the opensource web analytics software Piwik [15]. Our plugin allows the publisher to view the queries, number of answers and results as well as enables the DP noise process and forwarding of answers. In total, the publisher software is about 450 lines of PHP code.

The **data aggregator software** is a simple program that enables the publisher to upload the encrypted answers. The aggregator then decrypts and aggregates the answers, adds noise and returns the signed results to the publisher. Our implementation is about 275 lines of Java and PHP code.

6.2 Example scenario

We analyze the computational and bandwidth overhead we impose on the components via some micro benchmarks. Lacking information about current aggregators' infrastructure makes a comparison difficult. Nevertheless, to analyze our system's overhead, we use the following scenario. Each week, a publisher poses queries shown in Table 1 to 50K clients. The first eight queries collect the same information current aggregators provide to publishers. The last three are additional queries our system enables the publisher to pose: a 10-bucket histogram of the total number of pages visited by users across all sites, a 3-bucket histogram of visit frequency to each of 1000 websites selected by the publisher, and how many users use each of the top 5 search engines. We assume the aggregator uses a 2048-bit key.

Table 2: Per week bandwidth usage of the publisher and the data aggregator.

	Publisher	Data Aggregator
Collecting answers	0.37GB	-
Forwarding noise answers	1.20GB	1.20GB
Forwarding all answers	1.57GB	1.57GB

Table 3: Clients having used search engines on a given day. Actual/Publisher/Data Aggregator

Day	Google	Yahoo	Bing	None/Other
01/18	72/73/73	20/20/19	1/-1/10	44/42/49
01/19	63/57/59	20/21/20	2/4/2	29/29/29
01/20	54/57/52	17/18/17	0/-1/-3	29/30/30
01/21	59/62/59	16/15/15	0/5/4	30/29/34

6.2.1 Computational Overhead

To measure the computational overhead, we ran our client on a laptop running Mac OS X 10.6.8 on an Intel Core 2 Duo 2.66 GHz as well as on a smartphone running Android 2.3.5 with a 1 GHz processor. Our JavaScript client can achieve about 380, 20, and 16 encryptions per second on Google Chrome, Firefox, and on the smartphone, respectively. Note that JavaScript can be slower than native code.

We ran the publisher and the aggregator software on a machine with 2GB of memory running Linux 2.6.38 kernel on an Intel Xeon two cores 2.4GHz. The publisher software can generate and encrypt around 7980 answers per second. In our scenario, the expected total number of additional answers (i.e., $b \times o$) for all 11 queries is around 4.9M, taking the publisher less than 11 minutes per week to generate.

The data aggregator software can decrypt and aggregate about 270 messages per second. In our scenario, the aggregation takes about 3.6 hours per week for the first 8 queries and about 6.6 hours per week for all 11 queries. Most of this overhead is due to the additional noise answers.

6.2.2 Bandwidth Overhead

The compressed size of the biggest query (i.e., 5002 buckets) is about 35KB. In comparison, nytimes.com's homepage is about 500KB, excluding advertisements. Furthermore, buckets may not change very often, and can be cached.

The client's bandwidth overhead is in the order of a few kilobytes for sending responses. In our example, a client would consume about 8KB/week for all 11 queries. Table 2 shows the publisher's and the data aggregator's total bandwidth consumption per week. Most bandwidth consumption is related to the noise answers; however, the overhead is still acceptable: distributing nytimes.com's homepage to the 50K user sample *just once* would consume about 23.8GB.

6.3 Deployment

To test our system's feasibility, we deployed our client via our friends and mturk.com for 15 days with 236 unique clients. We report on their browsing activities. On average, there were 118 active clients daily. Each day, we queried clients about how many pages they browsed, which sites they visited, their visit frequency to these sites, and which search engines they used. We used 3K most popular sites from Alexa and set $\epsilon_P = \epsilon_{DA} = 0.5$. Note that our goal was to gain experience rather than gather meaningful data.

The clients in our deployment were fairly active; almost half of them having visited at least 100 pages. Major sites, such as google.com, youtube.com and facebook.com, were (as expected) reported more than many other sites. We also gathered some data on the usage frequency of these sites. Many users have visited google.co.in much more frequently than facebook.com or youtube.com. Table 3 shows the number of clients having used a search engine. Our client covered only three search engines and might have missed searches on other sites with search functionality (e.g., Wikipedia).

7. RELATED WORK

Although web analytics, as far as we know, has never been considered in a privacy context, there have been a number of approaches for providing users with privacy guarantees in distributed settings. Here, we review past work most related to our system, in the areas of anonymous communication, privacy-preserving aggregation and differential privacy.

Users can use a VPN proxy or an anonymizing network like TOR [22] for anonymous communication. While providing privacy benefits for anonymous browsing, these systems are not suitable for non-tracking web analytics: they may violate our non-tracking goal (e.g., a VPN proxy observing the source and the destination of a session), mislead the publisher to collect incorrect information (e.g., the proxy's or TOR exit node's address misleading a publisher using IP geolocation), or most importantly, do not provide any differential privacy guarantees (e.g., if sensitive data is collected).

For these reasons, researchers have proposed systems that both preserve users' privacy and enable accurate collection and aggregation of private information [45, 46]. These systems, however, either make assumptions about the collected information (i.e., that it will not to leak the source identity) [46], or require an algorithm to decide which data are safe to contribute, which may not be easy to devise [45]. In contrast, our system combines differential privacy and separate encryption of answer messages to protect against identity leakage through the data, without any assumptions or prerequisites. Furthermore, these systems rely on an anonymity network, such as TOR, to hide the source identity (i.e., IP address), whereas our system utilizes an already existing entity (i.e., the publisher) as an anonymizing proxy.

Applebaum et al. proposed a system more similar to ours, in which participants send their private data (i.e., {key, value} pairs) to a proxy, which then forwards these data to a database responsible for aggregation [27]. The system's main goal is to achieve this aggregation without exposing participants' keys to each other. This goal requires a strong cryptographic model, causing each participant higher overhead than in our system. Therefore, this system is perhaps more suitable for publishers wanting to share their own, already aggregated analytics data rather than for users.

While these systems provide users with *some* privacy guarantees, they do not adhere to differential privacy, which is considered to give stronger and more formal guarantees than existing techniques [34, 35, 37]. Many original uses of differential privacy, however, assume the existence of a central database controlling the disclosure of results [28, 41]. Although attempts have been made to provide differential privacy in a distributed environment, these attempts either incur high overhead [36] or suffer from client churn [47, 50], making them impractical in a large-scale environment.

To tackle this practicality problem, recent proposals employ different approaches for generating differentially-private noise. Duan et al. utilize two honest-but-curious (HbC) *servers to add noise*, and guarantee accuracy via relatively efficient, but still costly zero-knowledge proofs [33]. Götz and Nath also use two HbC servers, but propose that *users add noise*, such that honest users compensate for the noise that unavailable or malicious users did not generate [39]. While preserving honest users' privacy, this system also allows a malicious user to distort the result arbitrarily.

More recently, Chen et al. proposed a proxy-based system (PDDP) for achieving differential privacy in a distributed environment [32]. PDDP utilizes *only one* HbC proxy that distributes an analyst's queries to clients, collects responses, and adds noise in a *blind* fashion (i.e. does not know how much noise it added). PDDP does not scale for a web analytics application, for two reasons. First, PDDP has no way of selecting classes of users to receive a given query (i.e. all users that visit a given website). Our system exploits publishers for that purpose. Second, PDDP encrypts every bucket answer, 'yes' and 'no' alike, making it very costly for the large-bucket queries that are needed in web analytics.

8. CONCLUSION & FUTURE WORK

We present what is to our knowledge the first system for collecting accurate, extended web analytics without tracking users. Our system provides users with differential privacy guarantees, while not requiring new organizational components. It may be possible to apply our technique to other analytics problems, such as application analytics (e.g., mobile) and surveys about sensitive topics (e.g., elections, drug use). These scenarios, however, present additional constraints and challenges (e.g., developers without a website). We plan to examine them in more detail.

While our design avoids the need for a new HbC organizational component (e.g., a proxy), it does so at the cost of certain new threats (e.g., publisher dropping responses) and additional mechanisms. Even with an HbC proxy, however, the threat of isolation attacks through SQL or bucket manipulations remains. One avenue of future work is to explore new designs addressing these issues while maintaining the scalability properties of the current system, and to understand the trade-off points better.

One approach to mitigating the isolation attacks through SQL or buckets might be to simply withhold results for buckets with low-values [27]. Another approach might be to have clients simply not answer repeat queries; however, this approach clearly results in a utility loss that needs to be better understood. Malicious publishers may also try to bypass such a mechanism via small variations in queries, essentially querying the same information with slightly different queries. Potential defenses may borrow ideas from information flow, each client tracking which piece of information it has exposed previously [29, 44, 49]. Other sophisticated approaches applied in centralized settings may help the aggregator and the publisher achieve better accuracy [28, 41]. One avenue of future work is to understand whether we can extend their usability to our distributed setting.

Acknowlegdments

The authors would like to thank Rose Hoberman, the sysnets group members at MPI-SWS, and our anonymous reviewers for their valuable feedback and suggestions.

9. REFERENCES

[1] Abine. http://www.abine.com/.
[2] Analytics Technology Web Usage Statistics. http://trends.builtwith.com/analytics. Aug 2, 2012.
[3] AW Stats - Free log file analyzer for advanced statistics (GNU GPL). http://awstats.sourceforge.net.
[4] BlueKai Consumers. http://bluekai.com/consumers_optout.php.
[5] BrightTag ONE-Click Privacy. http://www.brighttag.com/privacy/.
[6] Doubleclick Ad Planner by Google. https://www.google.com/adplanner/planning/site_profile#siteDetails?uid=rottentomatoes.com&geo=US. Aug 2, 2012.
[7] EasyPrivacy. http://easylist.adblockplus.org/.
[8] FatCow Web Hosting. http://www.fatcow.com.
[9] Ghostery. http://www.ghostery.com/.
[10] Google Public Policy Blog | Keep your opt-outs. http://googlepublicpolicy.blogspot.com/2011/01/keep-your-opt-outs.html.
[11] Internet Explorer 9 Tracking Protection Lists. http://ie.microsoft.com/testdrive/Browser/TrackingProtectionLists/faq.html.
[12] iPage Web Hosting. http://www.ipage.com.
[13] Lawsuit accuses comScore of extensive privacy violations. http://www.computerworld.com/s/article/9219444/Lawsuit_accuses_comScore_of_extensive_privacy_violations.
[14] Open Web Analytics. http://openwebanalytics.com.
[15] Piwik Web Analytics. http://piwik.org.
[16] Piwik Web Hosting. http://www.arvixe.com/piwik_hosting.
[17] Privacy Lawsuit Targets Net Giants Over 'Zombie' Cookies. http://www.wired.com/threatlevel/2010/07/zombie-cookies-lawsuit.
[18] Quantcast Clearspring Flash Cookie Class Action Settlement. http://www.topclassactions.com/lawsuit-settlements/lawsuit-news/920.
[19] Quantcast Opt-Out. http://www.quantcast.com/how-we-do-it/consumer-choice/opt-out/.
[20] Safari Adds Do Not Track Features. http://mashable.com/2011/04/14/safari-do-not-track.
[21] The Mozilla Blog | Mozilla Firefox 4 Beta, now including "Do Not Track" capabilities. http://blog.mozilla.com/blog/2011/02/08.
[22] Tor Project. https://www.torproject.org/.
[23] Traffic and Demographic Statistics by Quantcast. http://www.quantcast.com/rottentomatoes.com?country=US#!demo. Aug 2, 2012.
[24] Usage Statistics and Market Share of Traffic Analysis Tools. http://w3techs.com/technologies/overview/traffic_analysis/all. Aug 2, 2012.
[25] W3 - BlueKai Proposal for Browser Based Do-Not-Track Functionality. http://www.w3.org/2011/track-privacy/papers/BlueKai.pdf.
[26] M. Abe and E. Fujisaki. How to date blind signatures. In *Advances in Cryptology – ASIACRYPT '96*. 1996.
[27] B. Applebaum, H. Ringberg, M. J. Freedman, M. Caesar, and J. Rexford. Collaborative, Privacy-preserving Data Aggregation at Scale. In *PETS*, 2010.
[28] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, 2007.
[29] A. Birgisson, F. McSherry, and M. Abadi. Differential privacy with information flow control. In *PLAS*, 2011.
[30] T. Cao, D. Lin, and R. Xue. A Randomized RSA-based Partially Blind Signature Scheme for Electronic Cash. *Computers & Security*, 24(1), 2005.
[31] D. L. Chaum. Blind Signatures for Untraceable Payments. *Advances in Cryptology (CRYPTO)*, 1982.
[32] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards Statistical Queries over Distributed Private User Data. In *NSDI*, 2012.
[33] Y. Duan, J. Canny, and J. Z. Zhan. P4p: Practical large-scale privacy-preserving distributed computation robust against malicious users. In *USENIX Security Symposium*, pages 207–222, 2010.
[34] C. Dwork. Differential Privacy. In *ICALP*, 2006.
[35] C. Dwork. Differential Privacy: A Survey of Results. In *TAMC*, pages 1–19, 2008.
[36] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *EUROCRYPT*, pages 486–503, 2006.
[37] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*, pages 265–284, 2006.
[38] Z. Eslami and M. Talebi. A New Untraceable Off-line Electronic Cash System. *Electronic Commerce Research and Applications*, 10(1):59 – 66, 2011.
[39] M. Götz and S. Nath. Privacy-Aware Personalization for Mobile Advertising. In *Microsoft Research Technical Report MSR-TR-2011-92*, 2011.
[40] S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *NSDI*, 2011.
[41] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1-2), Sept. 2010.
[42] D. Kifer and B.-R. Lin. Towards an axiomatization of statistical privacy and utility. In *PODS*, 2010.
[43] A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas. Releasing search queries and clicks privately. In *WWW*, pages 171–180, 2009.
[44] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.
[45] A. Nandi, A. Aghasaryan, and M. Bouzid. P3: A Privacy Preserving Personalization Middleware for Recommendation-based Services. In *HotPETS*, 2011.
[46] K. P. N. Puttaswamy, R. Bhagwan, and V. N. Padmanabhan. Anonygator: Privacy and Integrity Preserving Data Aggregation. In *International Conference on Middleware*, 2010.
[47] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD Conference*, pages 735–746, 2010.
[48] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *NSDI*, 2012.
[49] A. Sabelfeld and A. Myers. Language-based information-flow security. *Selected Areas in Communications, IEEE Journal on*, 21(1), 2003.
[50] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS*, 2011.
[51] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, 2010.

APPENDIX

A. ANALYSIS OF RESAMPLED NOISE

In this appendix, we prove that the noise added by the publisher still adheres to differential privacy principles. In

Algorithm 1 ResampledNoise

procedure Noise(function f with b dimensions, noise magnitude λ, user input x, offset o)
2: **for** each dimension i of f **do**
do $\tilde{X}_i \sim \text{Lap}(\lambda)$
4: **while** $\tilde{X}_i < -o$
$O_i = f(x)_i + \lfloor \tilde{X}_i \rfloor + o$
6: **end for**
return $O_1, O_2, \ldots, O_b$
8: **end procedure**

particular, we prove that resampling the noise value when it exceeds the offset still provides (ϵ, δ)-differential privacy.

DEFINITION 1 (INDISTINGUISHABILITY [36]). An algorithm $\mathcal{A}$ is (ϵ, δ)-differentially private if for all user data x, x' differing in the data of a single user and for all subsets $\mathcal{O}$ of the output space Ω:

$$\Pr[\mathcal{A}(x) \in \mathcal{O}] \leq e^{\epsilon} \Pr[\mathcal{A}(x') \in \mathcal{O}] + \delta$$

THEOREM 1. *Consider a histogram query f with b buckets. For a user u let u_i denote the user's value for bucket i of f. We require u_i to be a non-negative integer bounded by A, i.e., $u_i \in \{0, 1, \ldots, A\}$ and $\sum_{i=1}^{b} u_i = A$. We denote by x the values for a fixed set of users. With*

$$o \geq \lambda \ln\left(\left(e^{\frac{A}{\lambda}} - 1 + \delta/(2A)\right) A/\delta\right)$$

and

$$\lambda \geq 2A/\epsilon$$

Algorithm 1, ResampledNoise *gives (ϵ, δ)-differential privacy.*

In our system, a user may send multiple answers with 'N/A' or 'null' as bucket IDs. Therefore, we need to use the general bound for the user's value for bucket i (i.e., $u_i \in \{0, 1, \ldots, A\}$ rather than $u_i \in \{0, 1\}$).

PROOF. We start by analyzing a variant of ResampledNoise, in which no rounding takes place and we do not add o in the end. This replaces Line 5 with $O_i = f(x)_i + \tilde{X}_i$. We show that this variant preserves (ϵ, δ)-differential privacy. The privacy guarantee of the original ResampledNoise follows immediately from the fact that any transformation (in this case, rounding) of a differentially-private output also preserves privacy [42].

Consider any neighboring inputs x, x' and any outputs $\mathcal{O}$. We define the subset of bad outputs $\mathcal{O}'$ to be the o, such that there exists an i with $f(x)_i \neq f(x')_i$ and $o_i \leq f(x)_i + A - o$. Let us compute the probability of creating a bad output for each i, such that $f(x)_i \neq f(x')_i$. By X_i, we denote a variable distributed according to $\text{Lap}(\lambda)$.

$$\Pr[\tilde{X}_i < -o + A] \tag{3}$$
$$= \Pr[X_i < -o + A | X_i > -o] \tag{4}$$
$$= \frac{\Pr[-o < X_i < -o + A]}{\Pr[X_i > -o]} \tag{5}$$
$$= \frac{\Pr[X_i < -o + A] - \Pr[X_i < -o]}{1 - 1/2e^{\frac{-o}{\lambda}}} \tag{6}$$
$$= \frac{1/2e^{\frac{-o+A}{\lambda}} - 1/2e^{\frac{-o}{\lambda}}}{1 - 1/2e^{\frac{-o}{\lambda}}} \tag{7}$$

Since $f(x), f(x')$ differ in at most $2A$ dimensions, we have that

$$\Pr[\mathsf{Noise}(x) \in \mathcal{O}'] \leq 2A \frac{1/2e^{\frac{-o+A}{\lambda}} - 1/2e^{\frac{-o}{\lambda}}}{1 - 1/2e^{\frac{-o}{\lambda}}}$$

We claim that

$$\Pr[\mathsf{Noise}(x) \in \mathcal{O}'] \leq \delta \tag{8}$$

Proof.

$$\Pr[\mathsf{Noise}(x) \in \mathcal{O}'] \leq \delta$$
$$\Leftrightarrow \frac{1/2e^{\frac{-o+A}{\lambda}} - 1/2e^{\frac{-o}{\lambda}}}{1 - 1/2e^{\frac{-o}{\lambda}}} \leq \delta/(2A)$$
$$\Leftrightarrow \frac{e^{\frac{-o+A}{\lambda}} - e^{\frac{-o}{\lambda}}}{1 - 1/2e^{\frac{-o}{\lambda}}} \leq \delta/A$$
$$\Leftrightarrow \frac{e^{\frac{+A}{\lambda}} - 1}{e^{\frac{o}{\lambda}} - 1/2} \leq \delta/A$$
$$\Leftrightarrow \left(e^{\frac{+A}{\lambda}} - 1 + \delta/(2A)\right) A/\delta \leq e^{\frac{o}{\lambda}}$$
$$\Leftrightarrow \lambda \ln\left(\left(e^{\frac{+A}{\lambda}} - 1 + \delta/(2A)\right) A/\delta\right) \leq o$$

The last inequality is true based on our assumptions stated in Theorem 1. To complete the proof, it suffices to show:

$$\Pr[\mathsf{Noise}(x) \in \mathcal{O} \setminus \mathcal{O}'] \leq e^{\epsilon} \Pr[\mathsf{Noise}(x') \in \mathcal{O} \setminus \mathcal{O}']$$

Let us start by considering a dimension i. We note that

$$\Pr[\mathsf{Noise}(x)_i = o_i] = \Pr[\tilde{X}_i = o_i - f(x)_i]$$
$$= \begin{cases} \Pr[X_i = o_i - f(x)_i | X_i \geq -o] & \text{if } o_i - f(x)_i \geq -o \\ 0 & \text{otherwise} \end{cases}$$

For $o \in \mathcal{O} \setminus \mathcal{O}'$ we have by definition of $\mathcal{O}'$ that either $f(x)_i = f(x')_i$ or $f(x)_i + A - o < o_i$.

In the first case, we have $Pr[o_i = \mathsf{Noise}(x)_i] = Pr[o_i = \mathsf{Noise}(x')_i]$. In the second case, we have that both $f(x)_i + A - o < o_i$ as well as $f(x')_i - o < o_i$ (since $|f(x) - f(x')| \leq A$). Thus,

$$\frac{Pr[o_i = \mathsf{Noise}(x)_i]}{Pr[o_i = \mathsf{Noise}(x')_i]} = \frac{\Pr[X_i = o_i - f(x)_i]}{\Pr[X_i = o_i - f(x')_i]}.$$

We have that for $o \in \mathcal{O} \setminus \mathcal{O}'$

$$\frac{\Pr[\mathsf{Noise}(x) = o]}{\Pr[\mathsf{Noise}(x') = o]} \tag{9}$$
$$= \frac{\prod_{i=1}^{b} \Pr[\tilde{X}_i = o_i - f(x)_i]}{\prod_{i=1}^{b} \Pr[\tilde{X}_i = o_i - f(x')_i]} \tag{10}$$
$$= \frac{\prod_{i:f(x)_i \neq f(x')_i} \Pr[X_i = o_i - f(x)_i]}{\prod_{i:f(x)_i \neq f(x')_i} \Pr[X_i = o_i - f(x')_i]} \tag{11}$$
$$= \frac{\prod_{i:f(x)_i \neq f(x')_i} 1/(2\lambda) e^{-|o_i - f(x)_i|/\lambda}}{\prod_{i:f(x)_i \neq f(x')_i} 1/(2\lambda) e^{-|o_i - f(x')_i|/\lambda}} \tag{12}$$
$$\leq \prod_{i:f(x)_i \neq f(x')_i} e^{1/\lambda(-|o_i - f(x)_i| + |o_i - f(x')_i|)} \tag{13}$$
$$= e^{1/\lambda \sum_{i:f(x)_i \neq f(x')_i} -|o_i - f(x)_i| + |o_i - f(x')_i|} \tag{14}$$
$$\leq e^{1/\lambda \sum_{i:f(x)_i \neq f(x')_i} |f(x)_i - f(x')_i|} \tag{15}$$
$$\leq e^{2A/\lambda} \tag{16}$$
$$\leq e^{\epsilon} \qquad \text{By } \lambda \geq 2A/\epsilon \tag{17}$$

Putting Equations (8, 17) together, we have that

$$Pr[\mathsf{Noise}(x) \in \mathcal{O}]$$
$$= \Pr[\mathsf{Noise}(x) \in \mathcal{O}'] + \Pr[\mathsf{Noise}(x) \in \mathcal{O} \setminus \mathcal{O}']$$
$$\leq \delta + e^{\epsilon} \Pr[\mathsf{Noise}(x') \in \mathcal{O} \setminus \mathcal{O}']$$
$$\leq \delta + e^{\epsilon} \Pr[\mathsf{Noise}(x) \in \mathcal{O}],$$

which completes the proof. □