Differentially-Private Software Frequency Profiling under Linear Constraints

HAILONG ZHANG, Fordham University, USA
YU HAO, Ohio State University, USA
SUFIAN LATIF, Ohio State University, USA
RAEF BASSILY, Ohio State University, USA
ATANAS ROUNTEV, Ohio State University, USA

Differential privacy has emerged as a leading theoretical framework for privacy-preserving data gathering and analysis. It allows meaningful statistics to be collected for a population without revealing "too much" information about any individual member of the population. For software profiling, this machinery allows profiling data from many users of a deployed software system to be collected and analyzed in a privacy-preserving manner. Such a solution is appealing to many stakeholders, including software users, software developers, infrastructure providers, and government agencies.

We propose an approach for differentially-private collection of frequency vectors from software executions. Frequency information is reported with the addition of random noise drawn from the Laplace distribution. A key observation behind the design of our scheme is that event frequencies are closely correlated due to the static code structure. Differential privacy protections must account for such relationships; otherwise, a seemingly-strong privacy guarantee is actually weaker than it appears. Motivated by this observation, we propose a novel and general differentially-private profiling scheme when correlations between frequencies can be expressed through linear inequalities. Using a linear programming formulation, we show how to determine the magnitude of random noise that should be added to achieve meaningful privacy protections under such linear constraints. Next, we develop an efficient instance of this general machinery for an important subclass of constraints. Instead of LP, our solution uses a reachability analysis of a constraint graph. As an exemplar, we employ this approach to implement differentially-private method frequency profiling for Android apps.

Any differentially-private scheme has to balance two competing aspects: privacy and accuracy. Through an experimental study to characterize these trade-offs, we (1) show that our proposed randomization achieves much higher accuracy compared to related prior work, (2) demonstrate that high accuracy and high privacy protection can be achieved simultaneously, and (3) highlight the importance of linear constraints in the design of the randomization. These promising results provide evidence that our approach is a good candidate for privacy-preserving frequency profiling of deployed software.

CCS Concepts: • **Software and its engineering → Dynamic analysis**; • **Security and privacy → Privacy-preserving protocols**.

Additional Key Words and Phrases: frequency profiling, differential privacy, program analysis

ACM Reference Format:
Hailong Zhang, Yu Hao, Sufian Latif, Raef Bassily, and Atanas Rountev. 2020. Differentially-Private Software Frequency Profiling under Linear Constraints. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 203 (November 2020), 24 pages. https://doi.org/10.1145/3428271

Authors' addresses: Hailong Zhang, Fordham University, USA, hzhang285@fordham.edu; Yu Hao, Ohio State University, USA, hao.298@osu.edu; Sufian Latif, Ohio State University, USA, latif.28@osu.edu; Raef Bassily, Ohio State University, USA, bassily.1@osu.edu; Atanas Rountev, Ohio State University, USA, rountev.1@osu.edu.

2475-1421/2020/11-ART203
https://doi.org/10.1145/3428271

1 INTRODUCTION

Differential privacy [Dwork 2006; Dwork et al. 2006] has emerged as a leading theoretical framework for privacy-preserving data gathering and analysis. In addition to a rich body of theoretical results [Dwork and Roth 2014; SIGACT/EATCS 2017], differentially-private analyses have transitioned to industry in companies such as Google [Erlingsson et al. 2014], Apple [Apple 2017] and Uber [Uber 2017], as well as to government agencies such as the U.S. Census Bureau [Dajan et al. 2017].

Differential privacy allows meaningful statistics to be collected for a population without revealing "too much" information about any individual member of the population. In the area of software profiling, this machinery allows profiling data from many users of a deployed software system to be collected and analyzed in a privacy-preserving manner. Such a solution is appealing to many stakeholders: software users; software developers; providers of software analysis infrastructures; government agencies that enforce consumer protections.

1.1 Challenges and Contributions

Contribution 1: Higher-accuracy differentially-private profiling. There have been some recent efforts to introduce privacy-preserving software profiling. We argue that the closest related work, by Zhang et al. [2020a], takes an unnecessarily restrictive view of the needed privacy protections for a wide class of profiling analyses, which leads to a solution with artificially-low precision for the population-wide statistics being gathered. Rather than providing privacy at the level of individual run-time events as done in this prior work, we observe that it is sufficient to define privacy protections at the level of frequency vectors that accumulate the results from all events. Based on this observation, the first contribution of our work is an approach that formulates a different notion of differential privacy protection for profiling information, and uses a different mechanism to achieve this protection. The proposed approach reports the user-level frequency information with the addition of random noise drawn from the Laplace distribution. Our experimental results show that *significantly higher accuracy* of profiling results can be achieved, *without reducing privacy protections* in any substantial way. This result has significant implications for the design of any infrastructure for differentially-private remote profiling of deployed software.

Contribution 2: Differentially-private profiling under linear constraints. The frequencies of different run-time execution events are closely correlated due to the intrinsic static structure of the underlying software system. The second contribution of our work is a novel re-definition of the privacy guarantees and the corresponding privacy mechanisms to account for such correlations. For example, suppose the structure of the software implies that, in any run-time profile, a linear function of the frequencies of several events will always be greater than or equal to the frequency of another event. Such relationships are commonly observed in software systems. We argue that differential privacy protections must account for such relationships; otherwise, a seemingly-strong privacy guarantee is actually weaker than it appears. Our experimental results quantify this key concern. Motivated by this observation, we propose a *novel and general differentially-private profiling scheme when correlations between frequencies can be expressed through systems of linear inequalities*. Based on a sample of frequency profiles, our approach solves a linear programming (LP) problem to determine the magnitude of random noise that should be added to achieve meaningful differential privacy protections under such linear constraints. To the best of our knowledge, no prior work has incorporated such domain-derived data constraints in the design of a differentially-private analysis.

Contribution 3: Efficient differentially-private scheme for simple constraints. Next, we develop an instance of this general machinery for a special but important subclass of constraints, of the form $f(a) \geq f(b)$. Here $f(\ldots)$ denotes the run-time frequency of an event. In our exemplar

problem, we use the frequencies of method executions in Android apps, and derive such constraints using static analysis of call graphs and control-flow graphs in the app code. We demonstrate that in this case the magnitude of random noise can be determined efficiently. Our solution employs a *reachability analysis of a constraint graph that encodes such constraints*. By design, this approach is equivalent to the general LP formulation, but in practice is much more efficient. We employ this approach to implement differentially-private method frequency profiling for Android apps.

Contribution 4: Experimental study of privacy/accuracy trade-offs. Any differentially-private scheme has to balance carefully two competing aspects: privacy and accuracy. We perform *an experimental study to characterize these trade-offs* in several scenarios. In particular, we (1) show that our proposed randomization achieves much higher accuracy compared to related prior work, (2) demonstrate that high accuracy and high privacy protection can be achieved simultaneously, and (3) highlight the importance of linear constraints in the design of the randomization. These promising results provide evidence that our approach is a good candidate for privacy-preserving frequency profiling of deployed software.

1.2 Summary

This effort is a significant step in the new but increasingly important area of *privacy-preserving software analysis*. The need for such analysis is driven by the fundamental tension between the privacy of software users and the needs of software developers and researchers. Differential privacy is a rigorous tool for exploring such trade-offs. The proposed approach is important both as a solution to a core software usage analysis and as a step in broader efforts to develop privacy-preserving analyses of deployed software. Ultimately, work on such topics contributes to the area of user privacy, in an environment where widespread data gathering is rapidly becoming the norm and privacy protections are becoming essential.

In addition to technical considerations, data collection from software executions raises ethical questions that should not be ignored by computer science researchers. In this technological and societal context, the competing interests of various stakeholders (e.g., software users, businesses, and governments) require agreed-upon general principles and concrete algorithmic tools. For certain problems, differential privacy can help develop such tools both theoretically and practically (e.g., via open-source data collection libraries). While not a panacea, differential privacy quantifies the loss of privacy and makes it easier to reach informed compromises. Motivated by these general observations, we develop a novel frequency-gathering schema that achieves differential privacy when linear constraints exist in the run-time frequency profiles, as well as experimental evidence of the effectiveness of this technique when applied to method frequency profiling.

2 OVERVIEW AND MOTIVATION

A software system deployed across a population of many users can be analyzed remotely for several reasons. For example, mobile app analytics frameworks such as Google Firebase Analytics [Google 2020b] and Facebook Analytics [Facebook 2020] are employed by many mobile app developers to gather data about the usage of app features. As other examples, remote run-time information can be utilized for performance analysis [Han et al. 2012; Nagpurkar et al. 2006], focused testing [Pavlopoulou and Young 1999], debugging [Liblit et al. 2003; Ohmann et al. 2017], and failure analysis [Clause and Orso 2007; Jin and Orso 2012]. In all such techniques, per-user data is gathered locally and then sent to an analysis server, which is under the control of the developers or someone working on their behalf—e.g., an infrastructure provider such as Google or Facebook. In this paper we focus on one key example of such data collection: *event frequency profiling*.

2.1 Frequency Profiling

Our problem statement abstracts the process of remote frequency profiling as follows. We consider a software system that is deployed on the remote machines of n users. A set of events $\mathcal{V}$ is defined by software developers before software deployment. The execution of the software instance at each user $i \in \{1, \ldots, n\}$ triggers at run time a trace of k *event instances* that are recorded by the analysis infrastructure and sent to the remote server for further analysis. Without loss of generality, assume that k is decided by developers ahead of time and is publicly known. Thus, we consider "one-shot" data collection that uses a window of observation of k event instances, and does not collect any other data. Generalizing to a scenario with continuous and unrestricted data collection and reporting requires more advanced privacy-preserving techniques [Dwork and Roth 2014, Section 12.3] and is left for future work.

Each of the k event instances is an instance of some event $v \in \mathcal{V}$. For example, v could be "the run-time execution entered method m" and there could be several instances of v among the k event instances. We will use $f_i(v)$ to denote the number of instances of event v in the run-time trace of user i. For convenience, we will shorten "event instance" to "event" when this does not create any ambiguities. The local information for user i can be thought of as a vector of local frequencies $\boldsymbol{f}_i \in \mathbb{N}^{|\mathcal{V}|}$, with each entry corresponding to some event v. The profiling problem we consider is to obtain estimates of population-wide frequency information—that is, to estimate $F(v) = \sum_i f_i(v)$ for every $v \in \mathcal{V}$, or equivalently, to estimate the vector $\boldsymbol{F} = \sum_i \boldsymbol{f}_i$.

As a concrete example, consider analytics frameworks for Android apps. In such apps, each run-time event is logged by calling certain APIs with an identifier of the event. For example, in Firebase Analytics [Google 2020b], which appears in close to half of a large set of analyzed apps [Exodus Privacy 2020], method `logEvent` is used for recording events. This method takes as parameters a string and a map that uniquely identify the event. When this logging method is invoked, the underlying analysis infrastructure issues an HTTP request to record the logged event to the remote analytics server.

2.2 Privacy-Preserving Profiling

There is a large body of work on various forms of profiling, starting from basic information such as the frequencies of nodes and edges in control-flow models [Ball and Larus 1994] and extending to much more sophisticated run-time properties. When a user releases such information to other parties—e.g., software developers or analysis infrastructure companies—there is a fundamental question about privacy. The profiling information can provide details about execution of sensitive software functionality (e.g., how often the user changes her login credentials). The collected data can be used for characterizing user habits and interests, which can then be combined with other sources of information for the purposes of behavior profiling. There is a growing trend of aggregating data from various sources to create rich knowledge about users—e.g., by cross-linking records from various data providers and by applying sophisticated data mining. Data anonymization is not sufficient to address this problem [Narayanan and Shmatikov 2008, 2009]. The user has no control over unethical business practices (e.g., selling the data to third parties) or unexpected uses of the data due to silent changes in end-user privacy agreements, subpoenas by law enforcement, or security breaches in which user data is stolen and then shared with malicious actors.

As multiple sources of data about a person can be combined in ways that cannot be anticipated at the time when a profiling technique is deployed, it becomes increasingly important to deploy profiling techniques that are designed with theoretical guarantees against unknown privacy threats. Differential privacy is such a theoretical framework. A key property of this approach is that it ensures privacy protections in the extreme setting where an adversarial entity has access to large

amounts of auxiliary data about the individual, and employs such data in ways that are unknown to the designers of the differentially-private data gathering. These strong properties make differential privacy an appealing target for the designers of privacy-preserving software analyses, and were the motivation for developing our frequency profiling approach.

Next, we outline the differentially-private version of our frequency profiling analysis. Without privacy, a user reports to the remote analysis server her local frequency information, encoded as a vector of local frequencies $\boldsymbol{f}_i$. The server then simply computes and reports the global frequency vector $\boldsymbol{F} = \sum_i \boldsymbol{f}_i$. With a differentially-private schema, the local frequencies are modified with the help of some randomization mechanism R. We consider randomizers $R : \mathbb{N}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|}$, where the frequency $f_i(v) \in \mathbb{N}$ of event $v \in \mathcal{V}$ for user i is transformed into a "noisy" frequency $R(f_i(v)) \in \mathbb{R}$. As discussed shortly, the noise being added is drawn from a non-discrete probability distribution, which is why the randomized data is not in the domain of integers.

The randomizer R is the same for all software users and is designed and embedded in the software implementation before the software is deployed. It is important to note that differential privacy assumes that a privacy adversary knows the exact design of R. For example, since the randomization is implemented in the code of the software that resides on a software user's machine, reverse engineering of this code could reveal the exact algorithm for R. Nevertheless, the randomizer should still provide the differential privacy guarantee (which is described shortly) even in this adversarial setting. Intuitively, by observing $R(\boldsymbol{f}_i)$ an external entity should *not* be able to produce high-confidence estimates of the true data $\boldsymbol{f}_i$.

Each user i rounds each element of $R(\boldsymbol{f}_i)$ to the nearest natural number and then reports the result to the analysis server. The server computes a vector $\hat{\boldsymbol{F}} \in \mathbb{N}^{|\mathcal{V}|}$ which estimates the vector of true global frequencies $\boldsymbol{F}$. The computation of $\hat{\boldsymbol{F}}$ depends on the randomization mechanism R. In the approach we propose, we can simply compute $\hat{\boldsymbol{F}} = \sum_i \lfloor R(\boldsymbol{f}_i) \rceil$ where $\lfloor \ldots \rceil$ indicates the rounding function.

3 FEASIBLE FREQUENCY VECTORS UNDER LINEAR CONSTRAINTS

In many program profiling problems, there are constraints on the run-time frequencies that are imposed by the static structure of the code. For example, consider the following code:

```
void m1() { if (...) { m3(); m3(); } }
void m2() { if (...) m3(); }
```

Further, assume that there are no other calls to `m3` in the entire program. From this code structure, we can conclude that $2f(\texttt{m1}) + f(\texttt{m2}) \geq f(\texttt{m3})$ for any run-time execution of this code. Here $f(\texttt{m})$ denotes the frequency of method `m` in the method-execution-frequency vector $\boldsymbol{f}$.

Linear Constraints on Run-time Frequencies. In this paper we focus on a class of commonly-occurring constraints that can be expressed as *linear inequalities* over the elements of the frequency vector $\boldsymbol{f}$. These inequalities are of the form $\boldsymbol{A}\boldsymbol{f} \geq \boldsymbol{b}$, where $\boldsymbol{A}_{m \times |\mathcal{V}|}$ is an integer matrix encoding m linear functions of $|\mathcal{V}|$ variables and $\boldsymbol{b}_{m \times 1}$ is an integer vector. Here we assume that $\boldsymbol{f}$ is an integer vector of dimensionality $|\mathcal{V}| \times 1$. For the example from above, $\boldsymbol{A} = \begin{pmatrix} 2 & 1 & -1 \end{pmatrix}$ and $\boldsymbol{b} = \begin{pmatrix} 0 \end{pmatrix}$.

Feasible Frequency Vectors. For our problem statement, not all frequency vectors are *feasible* at run time. Infeasible frequency vectors do not represent any run-time user behavior and will never be observed during software execution. For the code example shown above, there is no execution that can produce a vector $\boldsymbol{f}$ with $2f(\texttt{m1}) + f(\texttt{m2}) < f(\texttt{m3})$. We define this property as follows:

Definition 3.1 (Feasibility). A frequency vector $\boldsymbol{f}$ is *feasible* if $\boldsymbol{f} \geq 0$, $\boldsymbol{A}\boldsymbol{f} \geq \boldsymbol{b}$, and $\|\boldsymbol{f}\|_1 = k$.

Here $\|\boldsymbol{f}\|_1$ denotes the L_1 norm of a vector $\boldsymbol{f}$, i.e., $\sum_{v \in \mathcal{V}} |f(v)|$. As discussed later, the design of a differentially-private scheme must consider this notion of feasibility, and in particular the linear constraints that feasible vectors satisfy. In our approach these linear constraints will be used to determine the magnitude of random noise that needs to be added in the scheme to achieve differential privacy, as described in Section 4. Prior work by Zhang et al. [2020a] does not employ such constraints in the design of its randomization; instead, it only uses a restricted form of constraints in a post-processing step that affects accuracy but not privacy. In general, if the designers of the profiling analysis do not account for such constraints, the scheme is fundamentally flawed: the introduced noise would be insufficient to prevent a privacy adversary from drawing higher-confidence inferences than what the scheme is intending to achieve. For example, in our experiments presented in Section 7.4, we observe that the privacy guarantee of an analysis that ignores such constraints is actually $26.5\times$ weaker than what its design is trying to accomplish. Further discussion of the importance of using constraints in the analysis design is provided in Section 5.5.

4 THE DIFFERENTIAL PRIVACY GUARANTEE

Informally, the differential privacy guarantee is of the following form: for any possible output of a differentially-private data analysis, the probability this output was produced from real data a and the probability this output was produced from some "neighbor" data b are close to each other. Thus, just by observing the analysis output, an adversary cannot distinguish with high probability the case when the input was a from the case when the input was some neighbor b of a. This probabilistic *indistinguishability* guarantee is the essence of differential privacy: it ensures that high-confidence inferences cannot be drawn from the released data.

4.1 Indistinguishability

In the context of our problem, this property applies to any two neighbor feasible frequency vectors $\boldsymbol{f}$ and $\boldsymbol{f}'$ that could represent the local data of one software user. The "neighbor" relation will be defined shortly. Following the standard approach from the differential privacy literature [Dwork and Roth 2014], we define *ϵ-indistinguishability* as follows:

Definition 4.1 (ϵ-indistinguishability). Randomizer R achieves *ϵ-indistinguishability* if for any two neighboring feasible frequency vectors $\boldsymbol{f}, \boldsymbol{f}' \in \mathbb{N}^{|\mathcal{V}|}$ and for any set $O \subseteq \mathbb{R}^{|\mathcal{V}|}$ of possible outputs of the randomizer, the following holds:

$$\frac{\Pr[R(\boldsymbol{f}) \in O]}{\Pr[R(\boldsymbol{f}') \in O]} \leq e^{\epsilon}$$

Here $\Pr[\ldots]$ denotes the probability of an event. This definition should be interpreted as follows: for any randomizer output, the likelihood that the real data was $\boldsymbol{f}$ is close to the likelihood that the real data was $\boldsymbol{f}'$. Thus, $\boldsymbol{f}$ and $\boldsymbol{f}'$ are indistinguishable in a probabilistic sense from the point of view of an adversary who observes the randomizer output. Parameter ϵ defines the strength of this protection. Smaller values of ϵ provide stronger protection but necessitate more "noisy" randomization which leads to less accurate population-wide estimates. In practical applications, values of ϵ such as $\ln(9)$ have been used [Erlingsson et al. 2014].

4.2 Defining Neighbors

Recall that in our problem statement, for any feasible frequency vector $\boldsymbol{f}$ we have $\|\boldsymbol{f}\|_1 = k$. The L_1 distance between two such frequency vectors $\boldsymbol{f}$ and $\boldsymbol{f}'$ is $\|\boldsymbol{f} - \boldsymbol{f}'\|_1$ and this distance is in the set

$\{0, 2, 4, \ldots, 2k\}$. To normalize, we define the distance between two feasible frequency vectors as

$$d(\boldsymbol{f}, \boldsymbol{f}') = \frac{1}{2} \|\boldsymbol{f} - \boldsymbol{f}'\|_1 = \frac{1}{2} \sum_{v \in \mathcal{V}} |f(v) - f'(v)|$$

We can then define the *neighbors* of a vector $\boldsymbol{f}$ as

$$Neighbors(\boldsymbol{f}) = \{\boldsymbol{f}' \mid d(\boldsymbol{f}, \boldsymbol{f}') \leq \tau\}$$

Here threshold τ is used to define the extent of this neighborhood. Note the different role of parameters τ and ϵ. Using τ, we define for which pairs of vectors we aim to ensure indistinguishability. The next section discusses the implications of this choice. Once the notion of neighbors is defined, ϵ determines the desired strength of the indistinguishability between any such pair of neighbors.

4.3 Randomization Based on Laplace Mechanism

We next show how to design a randomizer $R : \mathbb{N}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|}$ that achieves ϵ-indistinguishability. Our approach is a direct application of a classic technique from differential privacy: it draws random noise from the Laplace distribution and adds it to the "raw" frequency vector. The Laplace probability distribution $Lap(b)$, parameterized by a scale parameter $b > 0$ and centered at 0, is defined by the probability density function $p(y|b) = \frac{1}{2b} \exp(-\frac{|y|}{b})$. This is a symmetric version of the exponential distribution. We define the following randomizer:

$$R(\boldsymbol{f}) = \boldsymbol{f} + \begin{pmatrix} Y_1 \\ \vdots \\ Y_{|\mathcal{V}|} \end{pmatrix}$$

where each $Y_j \sim Lap(b)$ is drawn independently from the Laplace distribution with some scale parameter b. As a direct corollary from standard results in differential privacy, this R achieves ϵ-indistinguishability as long as $b \geq \frac{2\tau}{\epsilon}$.

4.4 Randomization Based on Randomized Response

Several differentially-private data analyses (e.g., Bassily et al. [2017]; Erlingsson et al. [2014]; Wang et al. [2017]; Zhang et al. [2020a]) have employed as a basic building block in their construction the technique of *randomized response* [Warner 1965]. This technique was initially proposed in social sciences to survey information that could be sensitive. The essence of randomized response is that a user's data is represented as a binary vector, with each bit corresponding to an event of interest, and then bits are inverted with some probability. The resulting randomized vectors from many users are aggregated and calibrated to obtain population-wide estimates.

This machinery has been employed by Zhang et al. [2020a] to gather event frequency information from software executions. Each run-time event is considered as a binary vector of size $|\mathcal{V}|$ with a single bit equal to 1, corresponding to the observed event. Each such vector is then randomized. The privacy guarantee for this approach ensures indistinguishability between the real event trace and all other traces that differ from it by at most t events.

Such an approach may be suitable for scenarios where event-by-event reports are sent to the server—for example, for current mobile app analytics frameworks where the randomization has to be conducted immediately after an event is triggered. However, it is unnecessarily "noisy" for cases when only the frequency vector of the trace needs to be shared. Rather than randomizing each event (via randomized response) and then accumulating the results in a frequency vector, we propose to accumulate the actual frequencies first and then randomize the resulting vector (using the Laplace mechanism). Theoretically, the second approach provably achieves better accuracy by roughly a factor of $\sqrt{k}$; this observation is a consequence of well-known results in differential

```
1  class SignInActivity {
2    User user;
3    boolean validPasswords(String password, String repeatPassword) {
4      Analytics.logEvent("SignInActivity.validPasswords"); // e1
5      return !Utils.isEmpty(password) && isPasswordValid(password)
6          && !Utils.isEmpty(repeatPassword) && password.equals(repeatPassword);
7    }
8    void registration() {
9      Analytics.logEvent("SignInActivity.registration"); // e2
10     EditText v1 = ...;
11     EditText v2 = ...;
12     String password = v1.getText();
13     String repeatPassword = v2.getText();
14     if (validPasswords(password, repeatPassword)) {
15       user.setPassword(password);
16     }
17   }
18 }
19 class User {
20   String password;
21   void setPassword(String password) {
22     Analytics.logEvent("User.setPassword"); // e3
23     this.password = password;
24   }
25 }
```

Fig. 1. Code derived from the `mitula` app.

privacy [Dwork and Roth 2014, Section 12.1]. Our experimental results in Section 7 empirically confirm that significantly higher accuracy is indeed observed in practice.

It is important to note that profiling in which aggregated data is reported (i.e., a frequency vector) has several advantages, including better efficiency, some data obfuscation even without differential privacy (as the sequence of events is not revealed), and a more accurate differentially-private version. For many profiling tasks, the collection and reporting of such aggregated data is sufficient to achieve the goals of profiling clients.

Example. As a concrete example, consider the collection the frequencies of method invocations in mobile apps. Figure 1 shows a snippet of code derived from the `mitula` app, which has more than 1 million downloads according to Google Play store [Mitula 2020]. Class `SignInActivity` allows the user to register a new account or log in using an existing account. After signing in to the app, the user's password, along with other critical information such as email address and authentication token, is stored in an instance of the `User` class. As introduced in Section 2.1, method invocations can be recorded by calling APIs in analytics frameworks. Here, we use Firebase Analytics [Google 2020b] for demonstration. As highlighted in the figure, the app can be instrumented by inserting a call to the `logEvent` API at the entry of every method with the method's signature being a parameter, so that each method call at run time is recorded and sent to the analytics server. For simplicity, we drop other parameters of `logEvent` in this example. At run time, the frequency of e_1 is guaranteed to be greater than or equal to the frequency of e_2, since each invocation of `registration` involves a call to `validPasswords` in class `SignInActivity`. There are no additional constraints for the three events

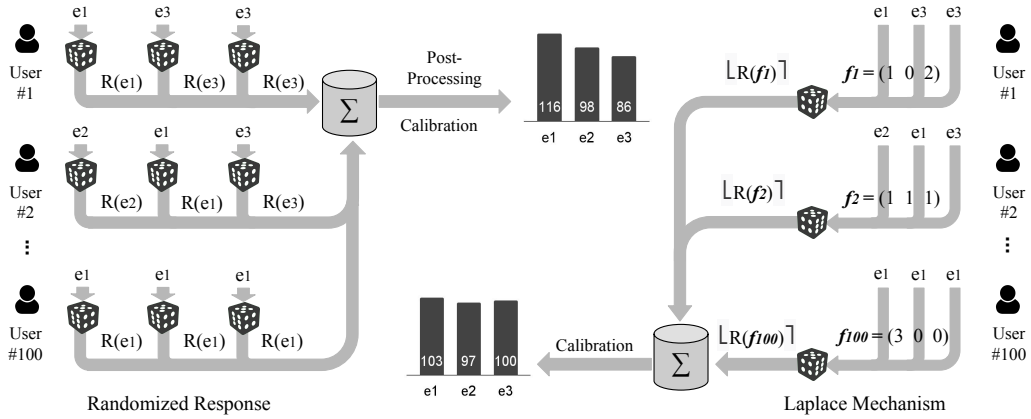


Fig. 2. Comparison between randomization based on randomized response and Laplace mechanism, by simulating 100 users using $\mathcal{V} = \{e_1, e_2, e_3\}$ from Figure 1, $k = 3$, $\epsilon = 1$, and $\tau = t = 1$.

as there are other call sites in the app of which the target is `SignInActivity.validPasswords` or `User.setPassword`.

Using the randomized-response-based approach, each call to `logEvent` is intercepted and the corresponding method-invocation event is perturbed. Figure 2 illustrates this process. It includes event traces from 100 simulated app users, derived from the actual data used in our experimental evaluation presented later in Section 7. Each trace records $k = 3$ run-time occurrences of the three events $\mathcal{V} = \{e_1, e_2, e_3\}$ in Figure 1. The actual aggregated frequency vector in this concrete example is $\boldsymbol{F} = \begin{pmatrix}105 & 88 & 107\end{pmatrix}$. We set $\epsilon = 1$ and $t = 1$ (number of differing events in neighbor traces), which defines the functionality of randomizer R. At the server, the randomization results for each event at each user are accumulated and post-processed to account for the added random noise. The aggregation result is then calibrated to produce the final frequency estimates. Details of this calibration step will be introduced shortly. As shown in the figure, the resulting vector of frequency estimates is $\begin{pmatrix}116 & 98 & 86\end{pmatrix}$.

The randomization based on Laplace mechanism is applied to local frequency vectors instead of individual events. When an event is triggered at a user i, it is accumulated locally in a frequency vector $\boldsymbol{f}_i$. The randomizer R directly adds noise drawn from the Laplace distribution to $\boldsymbol{f}_i$, as introduced in Section 4.3. To compare with the randomization based on randomized response, we use the same parameters for R, i.e., $\epsilon = 1$ and $\tau = 1$. As a result, the aggregated frequency vector in this concrete example is $\begin{pmatrix}103 & 97 & 100\end{pmatrix}$ after calibration. Its distance to the actual frequency vector $\boldsymbol{F}$ is 9, much smaller than the distance for the approach based on randomized response which is 21. Thus, the resulting estimates is "closer" to the true frequencies. This indicates that, at least for the events and settings in this example, the randomization based on Laplace mechanism produces more accurate analysis results for frequency profiling. In Section 7, we conduct extensive experiments on a set of apps and show the benefits of the proposed approach.

4.5 Calibration of Estimates

One problem of the random frequency perturbation performed by R is that it leads to *inconsistent* aggregated frequency vectors. As introduced in Section 3, every run-time frequency vector $\boldsymbol{f}$ satisfies the feasibility property, i.e., $\boldsymbol{f} \geq 0$, $\boldsymbol{A}\boldsymbol{f} \geq \boldsymbol{b}$, and $\|\boldsymbol{f}\|_1 = k$. It is desirable that the aggregated frequency estimate $\hat{\boldsymbol{F}} = \sum_i \lfloor R(\boldsymbol{f}_i) \rceil$ satisfies the constraints that would have been satisfied by the

true aggregated frequency $\boldsymbol{F} = \sum_i \boldsymbol{f}_i$. It is easy to see that those constraints are $\hat{\boldsymbol{F}} \geq 0$, $\boldsymbol{A}\hat{\boldsymbol{F}} \geq n\boldsymbol{b}$, and $\left\|\hat{\boldsymbol{F}}\right\|_1 = nk$. However, in general $\hat{\boldsymbol{F}}$ will violate the constraints. A standard approach to enforce such constraints is to calibrate the estimates in $\hat{\boldsymbol{F}}$. Following similar techniques from prior work [Wang et al. 2020; Zhang et al. 2020a], we compute a calibrated frequency estimate vector $\bar{\boldsymbol{F}} \in \mathbb{N}^{|\mathcal{V}|}$, which satisfies these constraints and minimizes its squared Euclidean distance to $\hat{\boldsymbol{F}}$. This calibrated estimate vector is the final output of the analysis.

Formally, we define the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{N}^{|\mathcal{V}|}} \quad & \left\|\boldsymbol{x} - \hat{\boldsymbol{F}}\right\|_2^2 \\ \text{subject to} \quad & \boldsymbol{x} \geq 0 \\ & \boldsymbol{A}\boldsymbol{x} \geq n\boldsymbol{b} \\ & \|\boldsymbol{x}\|_1 = nk \end{aligned}$$

The solution produced for $\boldsymbol{x}$ is the calibrated estimates $\bar{\boldsymbol{F}}$, which is the "closest" feasible vector to the aggregated frequency estimates $\hat{\boldsymbol{F}}$, in terms of squared Euclidean distance. Our implementation utilizes MATLAB's Optimization Toolbox [MathWorks 2020] to solve this optimization problem.

5 HIDING THE PRESENCE OR HOTNESS OF INDIVIDUAL EVENTS

One key question for the proposed randomization is how to select the value for τ. The notion of "protected distance" between frequency vectors ultimately has to be derived from some desired higher-level privacy properties. In this paper we consider two examples of such properties. First, we discuss a scenario where the analysis designer aims to ensure that the presence of a certain run-time event is hidden from a privacy adversary. More precisely, for some event $v \in \mathcal{V}$ that occurred at least once at run time, we would like to ensure that the adversary cannot distinguish, in the differential privacy sense, the actual run-time behavior from another possible behavior in which v was not observed at all. Equivalently, for any $\boldsymbol{f}$ with $f(v) > 0$, we would like to ensure that there exists at least one feasible $\boldsymbol{f}'$ such that $f'(v) = 0$ and $d(\boldsymbol{f}, \boldsymbol{f}') \leq \tau$.

A second scenario is where the analysis designer aims to hide from a privacy adversary the "hotness" of some event, by ensuring indistinguishability with vectors in which this event is "cold". In this scenario, an event whose frequency in $\boldsymbol{f}$ exceeds a certain threshold is considered frequently-executed (i.e., hot) in $\boldsymbol{f}$. Here for any $\boldsymbol{f}$ in which v is hot, we should ensure that there exists at least one feasible $\boldsymbol{f}'$ in which v is cold and $d(\boldsymbol{f}, \boldsymbol{f}') \leq \tau$. We next discuss the first scenario; the second one is presented later in this section.

We formulate the following problem: given a set $\mathcal{U} \subseteq \mathcal{V}$ of events whose presence in the run-time execution needs to be hidden, select the smallest distance threshold τ such that for any particular $v \in \mathcal{U}$, the presence of v is hidden in the sense discussed above. (A more precise formulation will be presented shortly in Section 5.1.) The choice of $\mathcal{U}$ depends on the usage scenario. For example, there may be some sensitive code functionality, e.g., changing a password, whose presence at run time should be hidden. Then $\mathcal{U}$ will contain any event related to these parts of the code. As another example, we could select a criterion of the form "at least $h\%$ of events are protected" based on some threshold h. In this paper we explore the second use case, but the underlying techniques are directly applicable to any other choices for $\mathcal{U}$.

Next we discuss the following key subproblem: given $\mathcal{U}$ and some feasible vector $\boldsymbol{f}$, determine the smallest distance threshold τ such that the presence of any particular $v \in \mathcal{U}$ in $\boldsymbol{f}$ is hidden by the randomization. Section 6 describes the general problem: how to select the value of τ to achieve privacy for the entire approach, over many software users i with different local vectors $\boldsymbol{f}_i$.

5.1 Difficulty of Hiding an Event

Consider any $\boldsymbol{f}$ with $f(v) > 0$. The presence of v in $\boldsymbol{f}$ can be hidden if $\boldsymbol{f}$ is indistinguishable from some neighbor $\boldsymbol{f'}$ with $f'(v) = 0$. We define the *difficulty of hiding v in $\boldsymbol{f}$* as the smallest distance at which such an $\boldsymbol{f'}$ exists:

$$D_v(\boldsymbol{f}) = \min_{\boldsymbol{f'}: f'(v)=0} d(\boldsymbol{f}, \boldsymbol{f'})$$

Here both $\boldsymbol{f}$ and $\boldsymbol{f'}$ must be feasible. If no feasible $\boldsymbol{f'}$ exists such that $f'(v) = 0$, it means that v always occurs at least once in all possible executions. In this case, it is impossible to provide protection to hide the presence of this v and we have $D_v(\boldsymbol{f}) = \infty$.

If we have a randomizer that uses a value of $\tau \geq D_v(\boldsymbol{f})$, the presence of v in $\boldsymbol{f}$ is hidden by the randomization. To achieve our goal of hiding any particular event from $\mathcal{U}$ in a given $\boldsymbol{f}$, this property has to hold for each $v \in \mathcal{U}$. In order to find the smallest such τ, we need to be able to compute $D_v(\boldsymbol{f})$ for any given v and $\boldsymbol{f}$. In the following subsection, for the general case of arbitrary linear constraints, we describe this problem as an integer linear programming problem. We then demonstrate that for the class of constraints of the form $f(v) \geq f(v')$, a more efficient alternative is to define a constraint graph and then perform analysis of this graph.

5.2 Computing Difficulty Using Linear Programming

We can compute $D_v(\boldsymbol{f})$ for any given $\boldsymbol{f}$ and v by solving the following optimization problem: minimize $\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{f}\|_1$ for $\boldsymbol{x} \in \mathbb{N}^{|\mathcal{V}|}$, $\boldsymbol{x} \geq 0$, $\boldsymbol{Ax} \geq \boldsymbol{b}$, $\|\boldsymbol{x}\|_1 = k$, and $x(v) = 0$. Here $\boldsymbol{x}$ is the unknown variable representing $\boldsymbol{f'}$ and the objective is to minimize the distance between $\boldsymbol{x}$ and $\boldsymbol{f}$. All constraints except for the last one ensure that $\boldsymbol{x}$ is a feasible frequency vector. This problem can be transformed into an *integer linear programming* (ILP) problem:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{s} \in \mathbb{N}^{|\mathcal{V}|}} \quad & \tfrac{1}{2}^{\mathsf{T}} \boldsymbol{s} \\ \text{subject to} \quad & \boldsymbol{x} \geq 0 \\ & \boldsymbol{Ax} \geq \boldsymbol{b} \\ & 1^{\mathsf{T}} \boldsymbol{x} = k \\ & x(v) = 0 \\ & \boldsymbol{s} \geq \boldsymbol{x} - \boldsymbol{f} \\ & \boldsymbol{s} \geq \boldsymbol{f} - \boldsymbol{x} \end{aligned}$$

The last two constraints ensure that $\forall v \in \mathcal{V}$ we have $|x(v) - f(v)| \leq s(v)$ and thus $\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{f}\|_1 \leq \frac{1}{2}^{\mathsf{T}} \boldsymbol{s}$. Therefore, the optimal value for the objective function gives us the desired value of $D_v(\boldsymbol{f})$.

5.3 Computing Difficulty Using Constraint Graph Analysis

Next, we focus on the special case of linear constraints of the form $f(v) \geq f(v')$. Such constraints can naturally occur when the static structure of the code enforces properties such as "the only way event v' could occur is if it is caused by event v".

To illustrate such constraints, consider method frequency profiling in a language such as Java. In this case $\mathcal{V}$ contains all methods m in the program code (excluding library/framework methods). Let $f(m)$ denote the number of times method m was executed at run time. We borrow the following exemplar static analysis from prior work [Zhang et al. 2020a]. First, if a call site in a caller method m' has only one possible target m, and this call site dominates the exit of m', then the frequency of m' is no greater than the frequency of m. Second, if there is a single call site in the program that may invoke a method m, this call site is not in a control-flow-graph loop, and m does not override any method from standard libraries or relevant frameworks (e.g., the Android framework), the

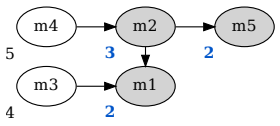


Fig. 3. Constraint graph for $\boldsymbol{f} = \begin{pmatrix}2 & 3 & 4 & 5 & 2\end{pmatrix}$, where nodes are labeled with frequencies in $\boldsymbol{f}$ and reachable nodes from m2 are marked in gray.

frequency of m' is greater than or equal to the frequency of m. For example, consider the following code:

```
void m1() { m2(); do { m3(); } while (...); }
void m2() { m4(); if (...) m5(); }
```

Suppose there are no other calls to m5 in the entire program, and m5 does not override library/framework methods. We can conclude that the following constraints hold: $f(\texttt{m2}) \geq f(\texttt{m1})$, $f(\texttt{m3}) \geq f(\texttt{m1})$, $f(\texttt{m4}) \geq f(\texttt{m2})$, and $f(\texttt{m2}) \geq f(\texttt{m5})$. As another illustration, for the code in Figure 1 we can conclude that $f(e_1) \geq f(e_2)$.

Constraint Graph Analysis. Each constraint $f(v) \geq f(v')$ can be encoded as an edge $v \rightarrow v'$ in a directed graph $G = (\mathcal{V}, \mathcal{E})$. Given this *constraint graph*, the computation of difficulty $D_v(\boldsymbol{f})$ for any given v and $\boldsymbol{f}$ with $f(v) > 0$ is as follows. Let R_v be the set of nodes reachable from v in G, including v. In any $\boldsymbol{f}'$ with $f'(v) = 0$, all nodes in R_v must have frequency of 0. To compute $D_v(\boldsymbol{f})$ we need to find such an $\boldsymbol{f}'$ that minimizes $d(\boldsymbol{f}, \boldsymbol{f}')$. A lower bound on $d(\boldsymbol{f}, \boldsymbol{f}')$ is the sum of frequencies in $\boldsymbol{f}$ for the nodes in R_v (since all of them will become 0 in $\boldsymbol{f}'$). If there exists $w \notin R_v$ such that w does not have any predecessors in graph, this lower bound is tight. In this case, to construct $\boldsymbol{f}'$, we can increase the frequency of w by $\sum_{v' \in R_v} f(v')$, without affecting any other nodes. Theoretically, such a w does not necessarily exist in general cyclic graphs. However, in the examples we have seen from real code, there is always a non-trivial number of events whose frequency is not constrained from above (i.e., they do not have incoming edges), and therefore this assumption is always satisfied. Based on these observations, we compute $D_v(\boldsymbol{f}) = \sum_{v' \in R_v} f(v')$. While the same result could be achieved with the more general LP formulation presented earlier, this graph analysis is more efficient and easy to implement.

Example. For the earlier example, $\mathcal{E} = \{\texttt{m4} \rightarrow \texttt{m2}, \texttt{m2} \rightarrow \texttt{m1}, \texttt{m2} \rightarrow \texttt{m5}, \texttt{m3} \rightarrow \texttt{m1}\}$. Further, suppose that $k = 16$ and we are given a vector $\boldsymbol{f} = \begin{pmatrix}2 & 3 & 4 & 5 & 2\end{pmatrix}$, where the first element is the frequency of m1, the second one the frequency of m2, etc. Suppose we are interested in $D_{\texttt{m2}}(\boldsymbol{f})$. Since $R_{\texttt{m2}} = \{\texttt{m1}, \texttt{m2}, \texttt{m5}\}$, as shown in Figure 3, we have $D_{\texttt{m2}}(\boldsymbol{f}) = 7$, the sum of frequencies of the three nodes, and the corresponding neighbor vector $\boldsymbol{f}' = \begin{pmatrix}0 & 0 & 4 & 12 & 0\end{pmatrix}$ or $\boldsymbol{f}' = \begin{pmatrix}0 & 0 & 11 & 5 & 0\end{pmatrix}$. This means that to hide m2 in f we have to also account for the execution frequencies of m1 and m5. As another example, $D_{\texttt{m4}}(\boldsymbol{f}) = 12$ with $\boldsymbol{f}' = \begin{pmatrix}0 & 0 & 16 & 0 & 0\end{pmatrix}$.

5.4 Hiding the Hotness of an Event

The approach presented earlier can be adapted to provide a different form of privacy protection. Suppose we have defined two categories of events: "hot" and "cold". An event v is hot in $\boldsymbol{f}$ if $f(v) > \eta$ where η is a pre-defined threshold, possibly dependent on k and $|\mathcal{V}|$. Given a hot event v in some $\boldsymbol{f}$, we can ensure that an adversary cannot distinguish this situation from another possible situation in which v was cold in $\boldsymbol{f}$.

Suppose we are given some v and $\boldsymbol{f}$ with $f(v) > \eta$. The hotness of v in $\boldsymbol{f}$ can be hidden if $\boldsymbol{f}$ is indistinguishable from some neighbor $\boldsymbol{f}'$ with $f'(v) \leq \eta$. In this case, the *difficulty of hiding the*

hotness of v in $\boldsymbol{f}$ is the smallest distance at which such an $\boldsymbol{f}'$ exists:

$$D_v(\boldsymbol{f}) = \min_{\boldsymbol{f}':f'(v)\leq\eta} d(\boldsymbol{f},\boldsymbol{f}')$$

Here both $\boldsymbol{f}$ and $\boldsymbol{f}'$ must be feasible. If no feasible $\boldsymbol{f}'$ with $f'(v) \leq \eta$ exists, this means that v is always hot in all possible executions. The computation of this difficulty can be expressed as an integer linear programming problem similar to the one in Section 5.2, except that constraint $x(v) = 0$ is replaced by $x(v) \leq \eta$. For the class of constraints described in Section 5.3, the constraint graph analysis considers all hot nodes v' reachable from v and computes $D_v(\boldsymbol{f}) = \sum_{v'} (f(v') - \eta)$.

5.5 Importance of Constraints in Randomizer Design

In designing our randomization approach, it is critical to account for the linear constraints $\boldsymbol{Af} \geq \boldsymbol{b}$ that must be satisfied by any feasible $\boldsymbol{f}$. To illustrate this point, suppose that for a given $\boldsymbol{f}$ with $f(v) > 0$, we are interested in hiding the presence of v.

Example. Consider the example presented earlier: constraints $f(\texttt{m2}) \geq f(\texttt{m1})$, $f(\texttt{m3}) \geq f(\texttt{m1})$, $f(\texttt{m4}) \geq f(\texttt{m2})$, and $f(\texttt{m2}) \geq f(\texttt{m5})$, $k = 16$, and $\boldsymbol{f} = \begin{pmatrix}2 & 3 & 4 & 5 & 2\end{pmatrix}$. As discussed, $D_{\texttt{m2}}(\boldsymbol{f}) = 7$.

Now suppose that instead of considering $\boldsymbol{f}'$ for which $\boldsymbol{f}' \geq 0$, $\boldsymbol{Af}' \geq \boldsymbol{b}$, $\|\boldsymbol{f}'\|_1 = k$, and $f'(v) = 0$, the designer of the randomizer ignored the linear constraints (or was not aware of them) and instead considered $\boldsymbol{f}'$ with $\boldsymbol{f}' \geq 0$, $\|\boldsymbol{f}'\|_1 = k$, and $f'(v) = 0$. It is easy to see that in that case the conclusion would be that the difficulty of hiding m2 is $f(\texttt{m2}) = 3$. For example, vector $\boldsymbol{f}' = \begin{pmatrix}2 & 0 & 7 & 5 & 2\end{pmatrix}$ would be considered one legitimate neighbor, and the conclusion would be that $D_{\texttt{m2}}(\boldsymbol{f}) = 3$.

What are the implications of this outcome? Suppose that $\mathcal{U} = \{\texttt{m2}\}$; that is, the only event we want to hide is m2. To achieve this protection for m2, the appropriate selection is $\tau = 7$. However, if instead we chose $\tau = 3$ because we ignored the linear constraints, we added insufficient noise because we used $Lap(\frac{6}{\epsilon})$ to draw the random noise when in reality we should have used $Lap(\frac{14}{\epsilon})$. Equivalently, the "effective" value of ϵ was increased by a factor of $\frac{7}{3}$. Recall from Definition 4.1 that the role of ϵ is to bound the ratio of probabilities between possible neighbors, which determines the strength of the indistinguishability guarantee. In effect, this ratio is now increased by a factor of $e^{\frac{7}{3}} \approx 10.31$. For example, if the designer of the randomizer was intending for the ratio of "probability that the real data was $\boldsymbol{f}$" and "probability that the real data was $\boldsymbol{f}'$" to be bounded by 3, in reality it is only bounded by 31. This could make one of these alternatives much more likely than the other one, which significantly weakens the privacy guarantee of indistinguishability.

In the general case, if the linear constraints are ignored, the effective value of ϵ is increased by a factor of $\frac{D_v(\boldsymbol{f})}{f(v)}$ for an event v. In Section 7 we further quantify these effects and show that this increase could be substantial. Thus, to ensure that the differential privacy guarantee is indeed provided with the expected strength defined by ϵ, it is imperative to design the randomizer to account for the linear constraints among frequencies, which in turn are the result of inherent correlations among software elements. Otherwise, a privacy adversary who is aware of these correlations may be able to draw inferences with confidence much higher than what the analysis designer intended, due to the insufficient magnitude of random noise.

6 OVERALL DESIGN OF DATA COLLECTION

Given a set of events $\mathcal{U}$ to be protected and a frequency vector $\boldsymbol{f}$, one can select $\tau = \max_{v\in\mathcal{U}} D_v(\boldsymbol{f})$ to ensure privacy for each element of this set. However, in a differentially-private scheme the value of τ must be selected ahead of time, before the randomizer is embedded in the software and distributed to software users. As a general property, if any parameter of the randomizer (e.g., ϵ or

τ) depends on the local data $\boldsymbol{f}_i$ of an user i, the analysis is not differentially-private for that user. To address this problem, we employ a model with two groups of users: *opt-in users* and *regular users*.

Opt-in Users. Each opt-in user i sends to the server the set of pairs $\langle v, D_v(\boldsymbol{f}_i)\rangle$ for all v that are observed at run time. This does leak some information to the server, and in general does not provide differential privacy for that user. However, we consider this a reasonable compromise since the raw profile $\boldsymbol{f}_i$ is not shared with anyone; only its coarse-grain characterization via this set of pairs becomes known to the server. Given this information, for each v reported by users the server computes $\tau(v) = \max_i D_v(\boldsymbol{f}_i)$. The set of $\tau(v)$ is sorted in ascending order and then used to select a set of events $\mathcal{U}$ covering the first h% of this sorted list. The largest $\tau(v)$ in this set is selected as the final value of τ that will be used later by regular users. The intent behind this approach is to calibrate the privacy/accuracy trade-offs by selecting the value of h and then determining which h% events can be protected with the smallest possible τ (i.e., the highest possible accuracy).

Regular Users. After the server computes τ from the opt-in users, this value is embedded in the randomizer which is then distributed to the regular users. For any regular user i, $R(\boldsymbol{f}_i)$ is sent to the server and used to compute the calibrated frequencies $\bar{\boldsymbol{F}}$ described in Section 4.5. One detail is that it is possible for some $D_v(\boldsymbol{f}_i)$ to exceed the bound τ. In this case, the user's data for v is obtained with weakened privacy protection. Formally, the randomizer achieves ϵ'-indistinguishability, where $\epsilon' = \frac{D_v(\boldsymbol{f}_i)}{\tau}\epsilon$. An alternative would be to exclude the data for v for such users from the data collection. In this case, no data will be collected from these users and the only leaked information is "the user's local difficulty exceeds τ".

One assumption for this data collection is that the distributions of data from opt-in users and from regular users are similar, and thus τ values inferred from the opt-in data can be applied to the regular users. In our experiments we infrequently see regular users whose local difficulty exceeds τ. However, in general this does not have to be the case. One possible solution is to differentially-privately check the distribution of the opt-in data against the distribution of regular data, and to adjust the data collection in case a "drift" starts occurring. This presents interesting questions for further investigation.

Others have used similar partitioning of users to achieve differential privacy and better accuracy. In particular, an approach by Avent et al. [2017] provides centralized-differential privacy for opt-in users, where the users' raw data is visible to but protected by the analysis server. Our approach prevents the collection of raw profiles from opt-in users and only gathers from them the difficulty information. As discussed earlier, this does not satisfy differential privacy in general. Depending on the definition of interesting events $\mathcal{U}$ and the protection goal h, this release of information could be negligible. To provide formal differential privacy protection for data gathered from opt-in users, standard approaches could potentially be applied, e.g., the Laplace mechanism.

7 EVALUATION

To empirically evaluate the proposed approach and its performance for privacy protection of event presence and hotness, we use 15 Android applications that were also used by prior related work [Zhang et al. 2020a]. We simulate 1000 users interacting with each app using the Monkey tool [Google 2020c]. Specifically, we run Monkey independently 1000 times and regard each separate Monkey execution as triggered by one software user. During this process, we collect method execution frequency information as follows. We instrument the entry of every method in each app to record the run-time execution of every method and accumulate its frequency in a local frequency vector at each user. We stop a simulation when the total frequency in the vector reaches $k = 5 \times |\mathcal{V}|$. As a result, we get 1000 run-time profiles of method frequencies for each app.

Table 1. Experimental subjects.

App	Classes	Methods	k
barometer	378	2237	11185
bible	1087	5340	26700
dpm	271	1362	6810
drumpads	446	1903	9515
equibase	251	1975	9875
localtv	714	3055	15275
loctracker	197	837	4185
mitula	966	7172	35860
moonphases	165	716	3580
parking	370	1649	8245
parrot	1235	7433	37165
post	1087	5340	26700
quicknews	1087	5340	26700
speedlogic	77	265	1325
vidanta	1568	9242	46210

The instrumentation is based on Soot [Sable 2020]. The profile-gathering mechanism utilizes Android Debug Bridge [Google 2020a] to communicate with and manage multiple instances of emulators for user simulation. Given the collected profiles, we run all randomization on a local machine separately from the executions that gather run-time frequencies. This allows us to conduct each experiment for multiple trials to report rigorous statistical results that account for the randomness introduced by local randomizers [Georges et al. 2007]. Table 1 shows the details of the subjects used in our experiments. Column "Classes" and "Methods" list the number of application classes and methods, excluding several well-known third-party Android libraries, e.g., `dagger` and `okio`. The threshold k for each app is shown in column "k" in the table.

All benchmarks and code for the evaluation described in this section are released publicly at https://presto-osu.github.io/dp-freq-prof.

7.1 Comparison Between Two Randomizers

To compare the accuracy of the two approaches based on randomized response and the Laplace mechanism discussed in Section 4, we fix the privacy parameter $\epsilon = 1$, which has been used in evaluation of other differentially-private analyses [Bassily et al. 2017], and alter the value of threshold τ for defining neighbors. We consider $\tau \in \{10^0, 10^1, 10^2, k\}$ and compute the vectors containing calibrated frequency estimates for the two approaches. We use *normalized error* (NE, "error" for short) with respect to the ground-truth frequencies $\boldsymbol{F}$ to measure the accuracy of the estimated frequency vectors. The error is the normalized distance between $\boldsymbol{F}$ and a given vector $\boldsymbol{x}$: $NE_{\boldsymbol{F}}(\boldsymbol{x}) = \frac{1}{k} d(\boldsymbol{x}, \boldsymbol{F}) = \frac{1}{2k} \|\boldsymbol{x} - \boldsymbol{F}\|_1$. The worst-case value for the error is 1, e.g., when $\boldsymbol{F} = \begin{pmatrix} k & 0 \end{pmatrix}$ and $\boldsymbol{x} = \begin{pmatrix} 0 & k \end{pmatrix}$ for a $\mathcal{V}$ with two events.

Figure 4 shows the measurements of the error for the randomized-response-based and Laplace-mechanism-based randomization for different values of τ. Unless otherwise specified, each experiment in this and the following subsections is repeated 30 times with the mean and 95% confidence interval of the results reported. The confidence intervals are typically very small, and barely noticeable in the figures.

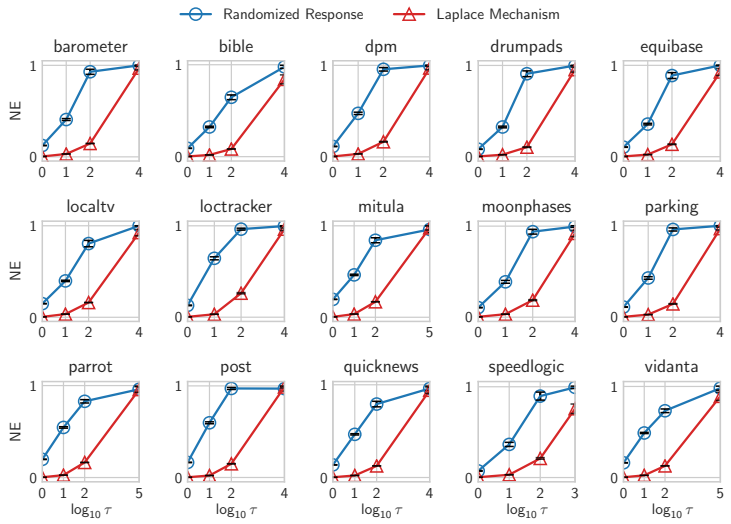


Fig. 4. Comparison of normalized error using randomization based on randomized response and on Laplace mechanism with $\epsilon = 1$.

Summary of results. As can be seen from these experiments, *randomization based on the Laplace mechanism outperforms the randomized-response-based approach in all cases*, especially for small values of τ. In particular, the error for the Laplace mechanism is 39.8× smaller than the error for the randomization based on randomized response when $\tau = 10^0$, 16.6× smaller for $\tau = 10^1$, and 5.9× smaller for $\tau = 10^2$, averaged across all apps. With the increase of τ, both randomizers require more extensive randomization in order to achieve differential privacy, and thus result in lower accuracy, which is to be expected.

Our conclusion is that instead of aiming for privacy protection for individual events, as done by Zhang et al. [2020a], a design for privacy at the level of frequency vectors via the Laplace mechanism achieves significant accuracy benefits. This observation is important for profiling frameworks for deployed software. It also raises interesting questions about potential redesign of widely-used software analytics frameworks such as Google Firebase Analytics [Google 2020b] and Facebook Analytics [Facebook 2020]. The current design of these frameworks provides logging mechanisms at the level of individual events, as illustrated by the code example in Section 4.4. It may be desirable to provide alternatives at a higher level of abstraction—e.g., a frequency vector that accumulates the effects of several events—since stronger privacy protections are possible at this case, and such protections could be implemented as optional functionality by the framework. Given the widespread use of these tracking frameworks in many thousands of popular mobile apps [Exodus Privacy 2020], this is a worthwhile direction of future investigations.

7.2 Hiding the Presence of Events

In this and the following subsection, we evaluate the accuracy of data collection and analysis introduced in Section 6 that allows the protection of each event from a set $\mathcal{U} \subseteq \mathcal{V}$, where $\mathcal{U}$ is selected depending on a threshold h and the difficulties reported by opt-in users. We first consider the goal of hiding the presence of individual events. Recall from Section 5 that each opt-in user i computes her difficulty $D_v(\boldsymbol{f}_i)$ for each event v such that $f_i(v) > 0$ using an analysis of the

Table 2. Average normalized error across all apps for hiding event presence.

h	Randomized Response			Laplace Mechanism		
	$\epsilon = 0.5$	$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 0.5$	$\epsilon = 1$	$\epsilon = 2$
25	0.422	0.316	0.220	0.027	0.012	0.007
50	0.642	0.510	0.394	0.069	0.039	0.021
75	0.873	0.765	0.628	0.148	0.097	0.065
100	0.996	0.994	0.989	0.917	0.793	0.646

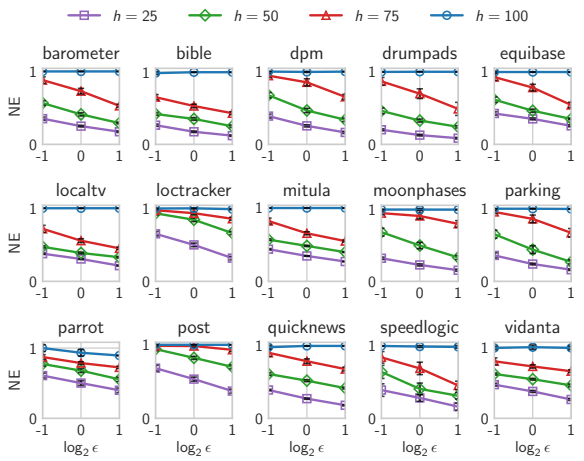
constraint graph. We run this computation on a machine with Xeon E5 2.2GHz CPU and 64GB RAM. The average cost of running the algorithm is 6.6 ms per user to calculate $D_v(\boldsymbol{f_i})$ for all qualified events v.

After collecting the difficulties from opt-in users, for each reported event v we compute $\tau(v) = \max_i D_v(\boldsymbol{f}_i)$ and set the final value of τ based on a threshold h such that *the presence of at least $h\%$ events is hidden*, i.e., τ covers the first $h\%$ of the sorted list of all $\tau(v)$ values. This τ is then used for the actual collection of frequency data from regular users. In the experiments, we consider threshold values $h \in \{25, 50, 75, 100\}$ to evaluate the accuracy of the proposed profiling under different protection goals. Higher threshold values provide protection for more events, lead to higher values for τ, and are expected to produce more error. To select opt-in users, we have used various settings and observed that using 10% of all users as the opt-in user group provides a reasonable τ and only a few users from the remaining 90% regular users have local difficulty exceeding this τ. Specifically, on average for each event, 10.5% users have difficulty $> \tau$ given $h = 25$. The percentage is 6.6% for $h = 50$, 2.9% for $h = 75$, and, of course, 0% for $h = 100$.

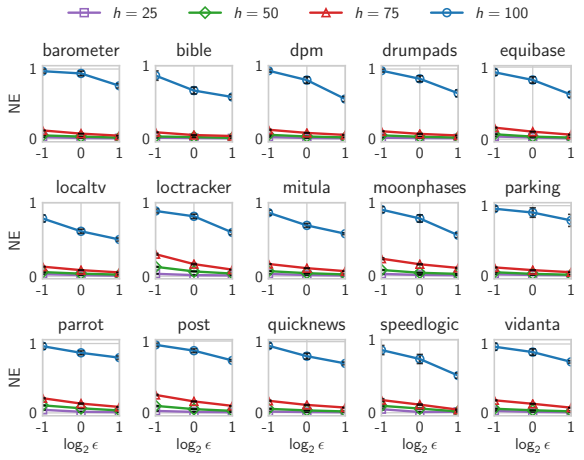
Table 2 shows the average error of the proposed approach and the approach in Zhang et al. [2020a] over all apps for different values of ϵ and h. We have used various values for ϵ in the experimental evaluation. Here we only show the results for $\epsilon \in \{0.5, 1, 2\}$ since we observe similar trends for other values. We can see that the error produced by the randomized-response-based approach is significantly larger than the error for the Laplace mechanism, regardless of the values of parameters. These results accord with the conclusion in Section 7.1.

For the proposed approach, *despite the inherent trade-offs between accuracy and privacy, high accuracy can be achieved simultaneously with a high protection goal (large h) and strong indistinguishability (small ϵ).* In particular, the error is below 0.1 if we want to protect the presence of at least a quarter or a half of all events, i.e., $h = 25$ and $h = 50$, even with very strong indistinguishability, e.g., $\epsilon = 0.5$. When the protection goal is raised to hide the presence of 75% of events, we still have relatively low error (≤ 0.15) for small values of ϵ. For example, the error is 0.097 when $\epsilon = 1$, which means that the calibrated frequency vector as a whole is only a few percentage points off the real vector. However, when $h = 100$, i.e., the protection goal is to protect the presence of all events, the error is above 0.5 for all choices of ϵ. This is to be expected, since some events have high frequencies across all users and large amounts of noise are needed to hide their presence.

Figure 5 shows more detailed measurements from these experiments. The analysis based on Laplace mechanism produces more accurate results for each app with larger ϵ since the indistinguishability strength in Definition 4.1 is reduced, leading to less noise. This conforms with the theoretical guarantee provided by the definition. As discussed before, when we enhance the protection to hide more events by increasing the value of h, there is more error in the resulting frequency estimate vector. For example, compared to $h = 25$, the error is $3.3\times$ larger for $h = 50$ and $8.9\times$ larger when $h = 75$, averaged across all apps for $\epsilon = 1$. If we want to protect the presence



(a) Randomized response



(b) Laplace mechanism

Fig. 5. Normalized error for hiding event presence with varying h and ϵ.

of all events, i.e., $h = 100$, the result is essentially unusable as the error is close to the worst-case value 1.

Summary of results. These experimental results are promising. Despite the fundamental tension between privacy (which requires higher magnitude of noise) and accuracy (which is reduced by this noise), we see evidence of a "sweet spot". In particular, our measurements demonstrate that high-accuracy estimates can be achieved together with privacy protection of the presence of most methods (i.e., high h) and strong differential privacy indistinguishability (i.e., low ϵ).

Table 3. Average normalized error across all apps for hiding event hotness.

h	Randomized Response			Laplace Mechanism		
	$\epsilon = 0.5$	$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 0.5$	$\epsilon = 1$	$\epsilon = 2$
25	0.428	0.313	0.220	0.042	0.022	0.012
50	0.635	0.510	0.394	0.099	0.059	0.033
75	0.870	0.757	0.626	0.221	0.143	0.089
100	0.996	0.995	0.993	0.919	0.798	0.646

7.3 Hiding the Hotness of Events

In this subsection we evaluate the effectiveness of the proposed approach for hiding the "hotness" of events. Recall from Section 5.4 that an event v is hot in a frequency vector $\boldsymbol{f}$ if $f(v) > \eta$ where η is a pre-defined threshold. Here we set $\eta = \frac{k}{|\mathcal{V}|}$. We have used other values of η and see similar results. As in the experiments from the previous subsection, we consider four definitions of set $\mathcal{U}$, based on a threshold h. For each reported event v we compute the largest value of the hotness-hiding difficulty $D_v(\boldsymbol{f}_i)$ across all opt-in users. We then set the final τ value to hide the hotness of at least $h\%$ of these events for $h \in \{25, 50, 75, 100\}$. A summary of the average error over all apps is shown in Table 3. The detailed results for each app are shown in Figure 6.

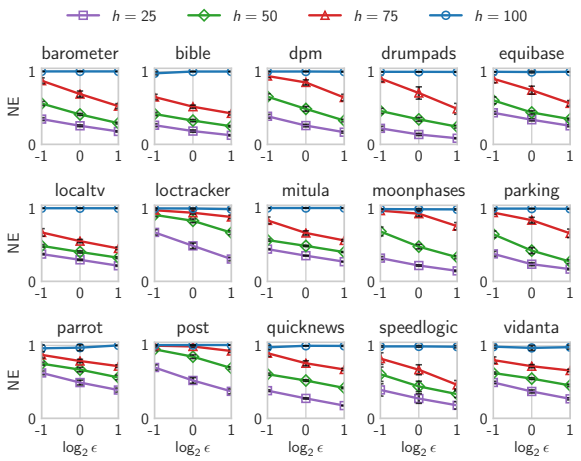
We can see that for the proposed approach based on Laplace mechanism, the error is low (< 0.1) for all values of ϵ used in the experiment when the protection goal is low, e.g., when we provide protection for half of the hot methods. When the value of h is increased to 75, the analysis can provide accurate results with somewhat larger values of ϵ. For example, we have error < 0.15 using $\epsilon = 1$, which is a typical value for this parameter in practice. At this protection level, the error is on average $2.4\times$ and $6.5\times$ larger than the error for $h = 50$ and $h = 25$, respectively. We reach the highest error when $h = 100$, similarly to the results from Section 7.2. This is due to the large frequencies of some extremely hot methods. One example of such a method is `DownloadProgress.onProgressUpdate` in the `dpm` app, which will be invoked multiple times to draw the progress bar when the user is downloading preset sound tracks. The randomizer has to introduce considerable amount of noise to hide the hotness of such methods.

Summary of results. The conclusions from these experimental results are similar to the ones from Section 7.2. For many (but not all) hot methods, our technique can effectively hide their hotness while at the same time providing high-accuracy estimates. Together with the earlier results for hiding event presence, this evaluation provides evidence of the promise of differentially-private analyses for remote profiling of deployed software.

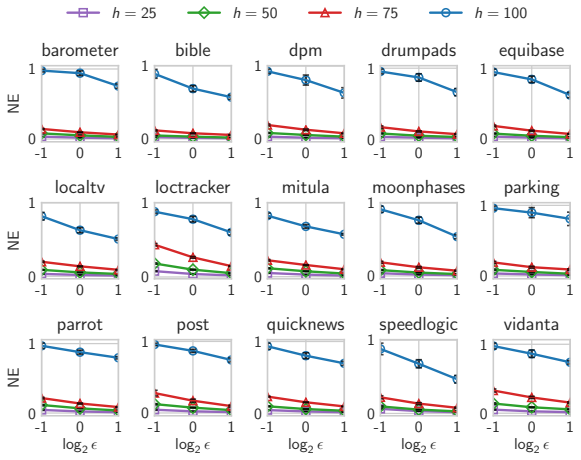
7.4 Implications of Enforcing Linear Constraints

Section 5.5 discussed the implications of including or excluding the linear constraints $\boldsymbol{A}\boldsymbol{f} \geq \boldsymbol{b}$ in the randomizer design. We empirically evaluated the effects on the privacy guarantee for these two scenarios and show the implicit degradation when the randomizer does not consider the constraints. For each of the two scenarios we computed the value of τ needed to hide the presence of individual events given $h = 25$, as was done for the previous experiments. (We have computed similar results for other values of h and observed even larger degradation.) We denote the two values by τ_{true} and τ_{false}, where the subscript indicates whether or not the constraints are taken into account during the computation of difficulties. We then calculate the degradation of the privacy guarantee as follows:

$$Degradation(\tau_{true}, \tau_{false}) = e^{\frac{\tau_{true}}{\tau_{false}}}$$



(a) Randomized response



(b) Laplace mechanism

Fig. 6. Normalized error for hiding event hotness with varying h and ϵ.

This definition shows the extent to which the bound on the ratio of the two probabilities in Definition 4.1 implicitly increases if the randomizer simply ignores the constraints. The mean value of this degradation for 30 independent repetitions of the same experiment is 26.5, averaged across all apps. This means that the probability bound is increased by a factor of 26.5 on average, by which the indistinguishability between neighboring frequency vectors is significantly weakened.

7.5 Effects of Randomness on Accuracy

As discussed earlier, we generate a method frequency profile by simulating user interactions with the app through a run of the Monkey GUI testing tool to trigger random actions. Such randomly generated profiles may not reflect the properties of real-world user data. In this subsection we

consider "less random" profiles and evaluate their effects on accuracy. In particular, we create and evaluate two subsets of the set of all 1000 frequency vectors, S_h and S_l, where the subscripts h and l denote *high-similarity* and *low-similarity*, respectively. Both sets are of size 500, but vectors in S_h are "closer" to each other in terms of distance than those in S_l. To construct the two subsets, we first compute the distance (defined in Section 4.2) between all pairs of frequency vectors. To compute S_h, we start with $S_h = \{\boldsymbol{f}_i, \boldsymbol{f}_j\}$ where $d(\boldsymbol{f}_i, \boldsymbol{f}_j)$ is the smallest among distances across all pairs of vectors. The next vector $\boldsymbol{f}_k$ is chosen such that the average distance of $S_h \cup \{\boldsymbol{f}_k\}$ is minimized. Here the average distance is defined as the average value of $d(\boldsymbol{f}, \boldsymbol{f}')$ for all $\boldsymbol{f}, \boldsymbol{f}' \in S_h$ such that $\boldsymbol{f} \neq \boldsymbol{f}'$. This process is repeated until we have $|S_h| = 500$. The subset S_l is created in a similar fashion: starting with $S_l = \{\boldsymbol{f}_i, \boldsymbol{f}_j\}$ where $d(\boldsymbol{f}_i, \boldsymbol{f}_j)$ is the largest among all pairs, we add $\boldsymbol{f}_k$ to S_l at each step such that $S_l \cup \{\boldsymbol{f}_k\}$ has the maximum average distance, until $|S_l|$ reaches 500. In our experiments, the average distance of S_l is 2.8× larger than that of S_h, on average across all apps.

We use the same metric and parameters as in Section 7.1 to evaluate how the randomness in profiles may affect the accuracy. The results show that the difference between the accuracy measurements for S_h and S_l is negligible for both the randomized-response-based and the Laplace-mechanism-based approaches. More specifically, averaged across the 15 apps, the error for randomized response using S_l is on average 0.95×, 1.09×, 1.04×, and 0.99× larger than the error using S_h under $\tau = 10^0, 10^1, 10^2$, and k, respectively. We observe similar results for the Laplace mechanism. This is to be expected. Since all randomized values have the same variance, with sufficiently large number of users the overall variance of the estimates should not depend on the actual data and its distribution in the domain of feasible frequency vectors. For privacy, the protection provides upper bounds on the privacy loss over the entire domain, regardless of the underlying distribution.

8 RELATED WORK

Differential privacy. Several examples of prior work on differential privacy were already discussed briefly [Bassily et al. 2017; Erlingsson et al. 2014; Wang et al. 2017; Zhang et al. 2020a]. We are not aware of any prior work that includes general linear constraints in the definition of neighbors and in the design of the randomizer. The closest related work by Zhang et al. [2020a] uses the randomized response technique and only embeds simple linear constraints in a calibration phase after randomization. Another related effort [Zhang et al. 2020b] considers node coverage in control-flow graphs and incorporates graph reachability constraints in the definition of privacy protection. In contrast, we consider local frequency counts and related linear inequalities. Many other analytics problems have also been considered in the differential privacy setting, including heavy hitters [Bassily et al. 2017; Bun et al. 2018], distribution estimation [Duchi et al. 2013], clustering [Nissim and Stemmer 2018], and learning [Kasiviswanathan et al. 2011]. These theoretical models have not yet been applied to software analysis and present a rich source of powerful privacy-preserving techniques. There also exist practical uses of differential privacy. Google's RAPPOR approach uses randomized response in the Chrome browser to gather data on popular URLs [Erlingsson et al. 2014]. Apple applies differential privacy when gathering various categories of user data [Apple 2017; Thakurta et al. 2017]. Microsoft uses differential privacy to collect device telemetry data [Ding et al. 2017]. Several of these existing approaches are focused on collection of a single data item per user, while our work aims at general profiling where accurate results are needed for both high-count and high-dimension local frequency profiles.

Privacy and programming languages. There is a growing number of efforts on differential privacy in the programming languages community. Zhang and Kifer [2017] propose LightDP, a relational type system, to verify sophisticated differential privacy algorithms requiring less annotations from programmers than customized logics based proofs. Inspired by LightDP, Wang

et al. [2019] further propose a flow-sensitive type system, ShadowDP, that uses shadow execution to verify more differential privacy algorithms and reduce the complexity of the verification process. Near et al. [2019] propose the Duet language for verifying differential privacy of general-purpose higher-order programs. Some emphasis has also been put on privacy in various fields of software engineering [Hadar et al. 2018]. Budi et al. [2011] propose to use kb-anonymity for testing and debugging data. The CLIFF+MORPH approach [Peters et al. 2013] uses perturbation to preserve data privacy of software defect information. With the exception of work already discussed, we are not aware of any efforts to create differentially-private versions of existing program analyses, which we believe is a fruitful target for future work.

Remote software profiling. Remote profiling of software has been studied extensively. Residual coverage monitoring [Pavlopoulou and Young 1999] collects statement coverage to aid post-deployment testing. The GAMMA system [Orso et al. 2002] tracks data from many users by assigning monitoring subtasks to different software instances and integrating results from probes to provide information about the original task. Diep et al. [2006] propose a probe distribution algorithm to collect profile events. Clause and Orso [2007] record program execution information, including interactions between software and environment and the input data for each interaction, for failure reproduction and in-house debugging. Ohmann et al. [2016] propose algorithms based on dominator trees to minimize the coverage probes required for monitoring. A related effort [Ohmann et al. 2017] introduces a new language for answering control-flow queries based on incomplete data from post-deployment failure reports.

Privacy in remote profiling has been the target of several projects, but none have used the machinery of differential privacy. One relevant approach [Elbaum and Hardojo 2004] considers anonymization of the collected data. Similarly, Clause and Orso [2011] anonymize and send back inputs that cause failures in deployed software instances, so that developers can debug the failures without knowing the actual inputs. There are no theoretical guarantees about the privacy protection provided by these techniques.

Many projects have used remote profiling data and could be interesting targets for development of differentially-private analyses. Orso et al. [2003] leverage GAMMA for impact analysis and regression testing using profiling and coverage data at block and method levels. Haran et al. [2005] use random forests to classify failing executions based on execution data. BugRedux [Jin and Orso 2012] collects execution data and synthesizes in-house executions the reproduce field failures. Follow-up work [Jin and Orso 2013] enhances BugRedux with debugging capabilities by synthesizing program executions that mimic a field failure based on call sequences. These approaches are interested in data such as counts of statements/functions, throw/catch counts, and execution traces. Our work, which provides well-defined privacy for frequency counters, presents a potential first step for developing differentially-private versions of these and similar analyses.

9 CONCLUSIONS

Differential privacy is a desirable theoretical framework for designing privacy-preserving software analysis, due to its inherently strong and rigorous privacy guarantees. When such machinery is applied to software profiling problems, or more generally to dynamic program analysis, it is important to consider domain-derived insights into the analyzed data. For the profiling problem we consider such data relationships are represented via linear constraints on frequencies. As we discuss theoretically and demonstrate experimentally, such constraints must be accounted for when designing a randomization scheme, in order to achieve the expected privacy protections.

For the specific problem of frequency estimation, we show that randomization of frequency vectors is more suitable than event-by-event randomization. This observation has implications for

the design of mobile app analytics frameworks such as Google Firebase Analytics and Facebook Analytics, as well we for any infrastructure for remote profiling of deployed software. Even with this design choice, it is still a challenge to select the parameters of the randomization in order to achieve desired trade-offs between accuracy and privacy. We show how to address this problem by defining and computing the difficulty of protecting certain events. Our results demonstrate that with careful application of this approach, it is possible to achieve both high privacy and high accuracy for the target goals of this work.

We anticipate that many interesting technical challenges will arise in attempts to apply differential privacy to other forms of dynamic program analysis. The work presented in this paper provides several potentially-useful building blocks for such future attempts.

ACKNOWLEDGMENTS

We thank the OOPSLA reviewers for their valuable feedback. This material is based upon work supported by the National Science Foundation under Grant No. CCF-1907715.

REFERENCES

Apple. 2017. Learning with privacy at scale. https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html.

Brendan Avent, Aleksandra Korolova, David Zeber, Torgeir Hovden, and Benjamin Livshits. 2017. BLENDER: Enabling local search with a hybrid differential privacy model. In *USENIX Security*. 747–764.

Thomas Ball and James Larus. 1994. Optimally profiling and tracing programs. *TOPLAS* 16, 4 (July 1994), 1319–1360.

Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. 2017. Practical locally private heavy hitters. In *NIPS*. 2288–2296.

Aditya Budi, David Lo, and Lingxiao Jiang. 2011. kb-anonymity: A model for anonymized behaviour-preserving test and debugging data. In *PLDI*. 447–457.

Mark Bun, Jelani Nelson, and Uri Stemmer. 2018. Heavy hitters and the structure of local privacy. In *PODS*. 435–447.

James Clause and Alessandro Orso. 2007. A technique for enabling and supporting debugging of field failures. In *ICSE*. 261–270.

James Clause and Alessandro Orso. 2011. Camouflage: Automated anonymization of field data. In *ICSE*. 21–30.

Aref Dajan, Amy Lauger, Phyllis Singer, Daniel Kifer, Jerome Reiter, Ashwin Machanavajjhala, Simson Garfinkel, Scot Dahl, Matthew Graham, Vishesh Karwa, Hang Kim, Philip Leclerc, Ian Schmutte, William Sexton, Lars Vilhuber, and John Abowd. 2017. The modernization of statistical disclosure limitation at the U.S. Census Bureau. https://www2.census.gov/cac/sac/meetings/2017-09/statistical-disclosure-limitation.pdf.

Madeline Diep, Myra Cohen, and Sebastian Elbaum. 2006. Probe distribution techniques to profile events in deployed software. In *ISSRE*. 331–342.

Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting telemetry data privately. In *NIPS*. 3571–3580.

John Duchi, Michael Jordan, and Martin Wainwright. 2013. Local privacy and statistical minimax rates. In *FOCS*. 429–438.

Cynthia Dwork. 2006. Differential privacy. In *ICALP*. 1–12.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *TCC*. 265–284.

Cynthia Dwork and Aaron Roth. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.

Sebastian Elbaum and Madeline Hardojo. 2004. An empirical study of profiling strategies for released software and their impact on testing activities. In *ISSTA*. 65–75.

Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *CCS*. 1054–1067.

Exodus Privacy. 2020. Most frequent app trackers for Android. https://reports.exodus-privacy.eu.org/en/reports/stats.

Facebook. 2020. Facebook Analytics. https://analytics.facebook.com.

Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically rigorous Java performance evaluation. In *OOPSLA*. 57–76.

Google. 2020a. Android Debug Bridge (adb). https://developer.android.com/studio/command-line/adb.

Google. 2020b. Firebase. https://firebase.google.com.

Google. 2020c. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.

Irit Hadar, Tomer Hasson, Oshrat Ayalon, Eran Toch, Michael Birnhack, Sofia Sherman, and Arod Balissa. 2018. Privacy by designers: Software developers' privacy mindset. *ESE* 23, 1 (2018), 259–289.

Shi Han, Yingnong Dang, Song Ge, Dongmei Zhang, and Tao Xie. 2012. Performance debugging in the large via mining millions of stack traces. In *ICSE*. 145–155.

Murali Haran, Alan Karr, Alessandro Orso, Adam Porter, and Ashish Sanil. 2005. Applying classification techniques to remotely-collected program execution data. In *ESEC/FSE*. 146–155.

Wei Jin and Alessandro Orso. 2012. BugRedux: Reproducing field failures for in-house debugging. In *ICSE*. 474–484.

Wei Jin and Alessandro Orso. 2013. F3: Fault localization for field failures. In *ISSTA*. 213–223.

Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SICOMP* 40, 3 (2011), 793–826.

Ben Liblit, Alex Aiken, Alice Zheng, and Michael Jordan. 2003. Bug isolation via remote program sampling. In *PLDI*. 141–154.

MathWorks. 2020. Optimization Toolbox. https://www.mathworks.com/help/optim.

Mitula. 2020. Mitula Homes. https://play.google.com/store/apps/details?id=com.mitula.homes.

Priya Nagpurkar, Hussam Mousa, Chandra Krintz, and Timothy Sherwood. 2006. Efficient remote profiling for resource-constrained devices. *TACO* 3, 1 (March 2006), 35–66.

Arvind Narayanan and Vitaly Shmatikov. 2008. Robust de-anonymization of large sparse datasets. In *S&P*. 111–125.

Arvind Narayanan and Vitaly Shmatikov. 2009. De-anonymizing social networks. In *S&P*. 173–187.

Joseph Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. 2019. Duet: An expressive higher-order language and linear type system for statically enforcing differential privacy. *PACMPL* 3, OOPSLA (2019), 1–30.

Kobbi Nissim and Uri Stemmer. 2018. Clustering algorithms for the centralized and local models. In *ALT*. 619–653.

Peter Ohmann, Alexander Brooks, Loris D'Antoni, and Ben Liblit. 2017. Control-flow recovery from partial failure reports. In *PLDI*. 390–405.

Peter Ohmann, David Bingham Brown, Naveen Neelakandan, Jeff Linderoth, and Ben Liblit. 2016. Optimizing customized program coverage. In *ASE*. 27–38.

Alessandro Orso, Taweesup Apiwattanapong, and Mary Jean Harrold. 2003. Leveraging field data for impact analysis and regression testing. In *ESEC/FSE*. 128–137.

Alessandro Orso, Donglin Liang, Mary Jean Harrold, and Richard Lipton. 2002. GAMMA system: Continuous evolution of software after deployment. In *ISSTA*. 65–69.

Christina Pavlopoulou and Michal Young. 1999. Residual test coverage monitoring. In *ICSE*. 277–284.

Fayola Peters, Tim Menzies, Liang Gong, and Hongyu Zhang. 2013. Balancing privacy and utility in cross-company defect prediction. *TSE* 39, 8 (2013), 1054–1068.

Sable. 2020. Soot – A framework for analyzing and transforming Java and Android applications. https://soot-oss.github.io/soot.

ACM SIGACT/EATCS. 2017. Gödel Prize. https://sigact.org/prizes/g%C3%B6del/citation2017.pdf.

Abhradeep Guha Thakurta, Andrew H Vyrros, Umesh S Vaishampayan, Gaurav Kapoor, Julien Freudiger, Vivek Rangarajan Sridhar, and Doug Davidson. 2017. Learning new words. In *Granted US Patents 9594741 and 9645998*.

Uber. 2017. Uber releases open source project for differential privacy. https://medium.com/uber-security-privacy/differential-privacy-open-source-7892c82c42b6.

Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally differentially private protocols for frequency estimation. In *USENIX Security*. 729–745.

Tianhao Wang, Milan Lopuhaä-Zwakenberg, Zitao Li, Boris Skoric, and Ninghui Li. 2020. Consistent and accurate frequency oracles under local differential privacy. In *NDSS*. 1–16.

Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. 2019. Proving differential privacy with shadow execution. In *PLDI*. 655–669.

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

Danfeng Zhang and Daniel Kifer. 2017. LightDP: Towards automating differential privacy proofs. In *PLDI*. 888–901.

Hailong Zhang, Yu Hao, Sufian Latif, Raef Bassily, and Atanas Rountev. 2020a. A study of event frequency profiling with differential privacy. In *CC*. 51–62.

Hailong Zhang, Sufian Latif, Raef Bassily, and Atanas Rountev. 2020b. Differentially-Private Control-Flow Node Coverage for Software Usage Analysis. In *USENIX Security*. 1021–1038.