A Hypothesis Testing Approach to Sharing Logs with Confidence

Yunhui Long
ylong4@illinois.edu
University of Illinois at Urbana-Champaign, USA

Le Xu
lexu1@illinois.edu
University of Illinois at Urbana-Champaign, USA

Carl A. Gunter
cgunter@illinois.edu
University of Illinois at Urbana-Champaign, USA

ABSTRACT

Logs generated by systems and applications contain a wide variety of heterogeneous information that is important for performance profiling, failure detection, and security analysis. There is a strong need for sharing the logs among different parties to outsource the analysis or to improve system and security research. However, sharing logs may inadvertently leak confidential or proprietary information. Besides sensitive information that is directly saved in logs, such as user-identifiers and software versions, indirect evidence like performance metrics can also lead to the leakage of sensitive information about the physical machines and the system.

In this work, we introduce a game-based definition of the risk of exposing sensitive information through released logs. We propose *log indistinguishability*, a property that is met only when the logs leak little information about the protected sensitive attributes. We design an end-to-end framework that allows a user to identify risk of information leakage in logs, to protect the exposure with log redaction and obfuscation, and to release the logs with a much lower risk of exposing the sensitive attribute. Our framework contains a set of statistical tests to identify violations of the log indistinguishability property and a variety of obfuscation methods to prevent the leakage of sensitive information. The framework views the log-generating process as a black-box and can therefore be applied to different systems and processes. We perform case studies on two different types of log datasets: Spark event log and hardware counters. We show that our framework is effective in preventing the leakage of the sensitive attribute with a reasonable testing time and an acceptable utility loss in logs.

CCS CONCEPTS

• **Security and privacy → Domain-specific security and privacy architectures**.

KEYWORDS

indistinguishability; hypothesis test; log obfuscation; privacy

ACM Reference Format:
Yunhui Long, Le Xu, and Carl A. Gunter. 2020. A Hypothesis Testing Approach to Sharing Logs with Confidence. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), March 16–18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3374664.3375743

CODASPY '20, March 16–18, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7107-0/20/03…$15.00
https://doi.org/10.1145/3374664.3375743

1 INTRODUCTION

Logs generated by systems and applications contain a wide variety of information such as timestamps, the size of input and output files, and CPU utilization. This information plays an important role in scientific research and analysis of production systems in industry. Although some of these analyses can be done internally, there has been a growing need to share logs with external analysts. For example, researchers rely on logs from real-life production systems to understand the workload and performance of these systems, and large companies hope to share this information so that they can guide academic work to be more relevant to their technical challenges [33]. On the other hand, small companies often outsource their log analysis process to third-party service providers such as Anomaly (anomaly.io) and Anodot (anodot.com) to benefit from a larger log-database and cutting-edge technologies.

Despite the increasing need for log sharing, companies are often unwilling to share their logs due to concerns about information exposure. Indeed, logs have the potential of revealing sensitive information about the system or program that generated it, ranging from applications, algorithms, to software and hardware configurations. For example, releasing traces of real-life production workloads may result in the leakage of information about new products if the prototypes of these products are using the same infrastructure [33].

Some sensitive information is directly saved into logs and can be protected by sanitization or anonymization. For example, several attempts have been made to anonymize IP addresses in network traces [48, 49]. However, log anonymization and sanitization is not enough to prevent the leakage of all the sensitive information. Since the metrics stored in logs are highly correlated with various properties of the system and hardware, seemingly nonsensitive metrics may reveal sensitive information. For instance, it is possible to infer information about a physical machine (e.g. the amount of RAM, the number of cores) if both workload information and performance metrics are released.

Differential privacy [12] has been shown to be effective in preventing side-channel leakages from timing and message sizes [50], but these studies only cover a portion of the information that is included in a log file. Meanwhile, achieving differential privacy on high-dimensional complex data with a reasonable amount of noise remains an open research problem [51].

In contrast to the extensive research on log anonymization, there is a lack of a systematic understanding of the sensitive information that can be indirectly inferred from a log file. Yet, this indirect leakage has been a great concern of companies in terms of releasing

production logs. To address this concern, researchers at Google proposed several obfuscation techniques [33], including sampling, scaling, and aggregation, to prevent the information leakage. However, there are two key limitations in these techniques. First, the obfuscation techniques are designed specifically to protect the traces from Google production clusters. The effectiveness of these techniques need to be generalized to different protection criterion, different systems, and different types of log files. Second, due to the lack of a clear protection criterion, there is no quantitative analysis of the effectiveness of the obfuscation techniques. Consequently, it is challenging for a user to understand the goal of each obfuscation techniques and to adapt them for their own systems.

Our Contributions. This paper aims to provide a framework that enables general-purpose log sharing under user-specified protection requirements. This framework consists of three major components: protection specification, indistinguishability tests, and log obfuscation.

First, we formalize the risk of revealing sensitive information during log sharing under the definition of *log indistinguishability*. We then enable users (log producers) to specify some sensitive attributes to be protected. We model the process of inferring the sensitive attributes from logs as a distinguishing game between the adversary and the user. The log indistinguishability property is met only if the adversary gets little or no advantage over random guessing in the distinguishing game. Log indistinguishability provides a formal protection criterion for our log-sharing framework. The remainder of the framework is designed around checking and ensuring this criterion in shared logs.

Second, we propose a set of *indistinguishability tests* to check whether a log file satisfies log indistinguishability. Similar to the randomness tests for Psuedo-Random Number Generators (PRNG), we use hypothesis tests to identify statistical patterns that may leak the sensitive information in a log file. We design a variety of tests to cover different data types.

Third, we propose log obfuscation techniques to help mitigate the information leakage identified by the indistinguishability tests. We design an iterative process between testing and obfuscation to help users strike a balance between protection of the sensitive information and the information loss incurred by obfuscation.

Finally, we evaluate our log-sharing framework under two case studies on different types of log datasets: Spark event logs and hardware performance counters. We show that our framework is effective in preventing the leakage of the sensitive attribute with a reasonable testing time and an acceptable information loss in logs.

Our contributions can be summarized as follows:

- We introduce a game-based definition of the risk for exposing sensitive information through log sharing.
- We design an end-to-end framework that allows users to identify risk of information leakage in logs, to protect the exposure with log obfuscation, and to release the logs with a much lower risk.
- With two case studies, we show that our framework is effective in preventing the leakage of the sensitive attribute with a reasonable testing time and an acceptable utility loss.

2 BACKGROUND

In this section, we briefly introduce the definition of differential privacy, its application to side channel protections, and the hypothesis tests that are used in this paper.

2.1 Differential Privacy

Proposed by Dwork et al., differential privacy [12] formalizes the vague concept of privacy into a provable property. We recall the definition of differential privacy.

Definition 2.1 ((ε, δ)-Differential Privacy). A randomized algorithm $\mathcal{M}$ is (ε, δ)-differentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for any neighboring inputs x_1 and x_2: $\Pr[\mathcal{M}(x_1) \in \mathcal{S}] \leq \exp(\varepsilon) \Pr[\mathcal{M}(x_2) \in \mathcal{S}] + \delta$. Specifically, when $\delta = 0$, $\mathcal{M}$ is ε-differentially private.

Differential privacy has been widely applied in dataset privacy. In most cases, one considers two datasets that differ by only one record as neighboring inputs, and differential privacy guarantees that no adversary can achieve a high advantage in inferring the membership of any record [20].

2.2 Hypothesis Tests

A *statistical hypothesis* is a testable statement about a process that is modeled via a set of random variables. The *null hypothesis*, denoted as $\mathcal{H}_0$, is the statistical hypothesis we want to reject, and the *alternative hypothesis*, denoted as $\mathcal{H}_A$, is the complement of the null hypothesis. A *hypothesis test* is the process of either accepting or rejecting the null hypothesis based on the observed data samples. A *two-sample test* is a hypothesis test performed on two data samples.

For example, suppose d_1 is a data sample obtained from distribution $\mathcal{D}_1$, and d_2 is a data sample obtained from distribution $\mathcal{D}_2$. In a two-sample test, the null hypothesis ($\mathcal{H}_0$) could be $\mathcal{D}_1 = \mathcal{D}_2$, and the alternative hypothesis ($\mathcal{H}_A$) could be $\mathcal{D}_1 \neq \mathcal{D}_2$. By performing the hypothesis test, we would like to determine whether d_1 and d_2 come from the same distribution.

The performance of a hypothesis test is evaluated by two types of errors. A type I error occurs if the test incorrectly rejects the null hypothesis, and a type II error occurs if the test incorrectly accepts the null hypothesis. In most hypothesis tests, it is more important to control type I errors. Therefore, a user would specify a significance level α (e.g. $\alpha = 0.01$ or $\alpha = 0.05$). The test rejects $\mathcal{H}_0$ if and only if the p-value, which is the output of the testing algorithm, is smaller than α. A correct hypothesis test needs to guarantee that, for all $0 \leq \alpha \leq 1$, we have $\Pr[p \leq \alpha \mid \mathcal{H}_0] \leq \alpha$. The *power* $(1 - \beta)$ of a hypothesis test is the probability that the test correctly rejects the null hypothesis. Under the same significance level α, tests with stronger power are more desirable because they have lower type II errors. Hypothesis tests can be divided into two categories: parametric tests and non-parametric tests. *Parametric tests* assume that the tested samples come from a known distribution, while *non-parametric tests* do not rely on this assumption.

Hypothesis tests are useful tools in statistical analysis, and a large number of hypothesis tests have been proposed to solve different problems. In this section, we briefly introduce three hypothesis tests that are used in this paper: χ^2 two sample tests, kernel two sample tests, and differential privacy tests. Both χ^2 tests and kernel

tests have the null hypothesis that two data samples come from the same distribution. χ^2 tests are suitable for categorical data or binned numerical data, while kernel tests are often performed on multidimensional numerical data. Differential privacy tests check whether an algorithm is differentially private under a given privacy budget ε.

Pearson's Chi-Squared Two-Sample Tests. A Pearson's chi-squared two-sample test (χ^2 test) [24] is performed on categorical data or binned numerical data. It tests the null hypothesis that the two samples come from the same distribution. The test is based on the assumption that the number of points in each bin should be similar if the null hypothesis is true.

Kernel Two-Sample Tests. The null hypothesis of a kernel two-sample test [16] is that the two samples come from the same distribution. The test statistic, *maximum mean discrepancy* (MMD) [38], is the largest difference in expectations over the output of a class of kernel functions.

Differential Privacy (DP) Tests. Given a mechanism M, two neighboring inputs x, y, and an output set S, a DP test [10] checks if M is ε-DP. Suppose $p_1 = \Pr[M(x) \in S]$, $p_2 = \Pr[M(y) \in S]$. The statistical hypotheses to be tested are

$$\mathcal{H}_0 : \quad p_1 \leq e^{\varepsilon} \cdot p_2, \qquad \mathcal{H}_A : \quad p_1 > e^{\varepsilon} \cdot p_2.$$

The p-value of the test is calculated based on the number of times $M(x)$ and $M(y)$ falls in S. The null hypothesis is rejected if $M(x)$ falls in S with a much higher frequency than $M(y)$.

3 LOG INDISTINGUISHABILITY

In this section, we formalize the risk of leaking sensitive information through shared logs. First, we introduce the log sharing problem and the adversary model used in this paper. Then, we propose a game-based definition to describe the risk of leaking the sensitive information. Finally, we provide an overview for our testing-based log obfuscation framework.

3.1 Problem Statement

Problem Setup. Suppose P is a log-generating process, and l is the log file produced by P. We study what information can be inferred about P by observing l. We assume l is parsed into a multidimensional time sequence vector consisting of numerical and categorical data. There have been extensive prior studies on parsing unstructured logs into structured sequential data [11, 17].

The sensitive information of P can refer to any information that is related to the computation process and not directly stored in the log file. This information may include software and hardware configurations, information about the physical machines (e.g. number of cores, the amount of RAM), and workload information such as algorithms and hyperparameters. In practice, the information that needs to be protected varies among applications. Therefore, we allow the users to specify the sensitive information that needs to be protected. The remaining information about P is considered to be nonsensitive and safe to release. The sensitive information can be a combination of different attributes of P. However, for simplicity, we view all the sensitive information as a single *sensitive attribute*, denoted by X, and X can be a vector of different configurations. Let $C = \{x_i \mid i \in \mathbb{Z}^+, i \leq M\}$ be a set of potential values for X known by the adversary. We call C the *candidate set* of X.

Adversary Model. We consider an external adversary that does *not* collocate with the target program P and has no control over P. The adversary can only infer information by analyzing the log files shared by the users, and users can sanitize or remove any sensitive information before sharing the log files. We also assume that the adversary has access to all the nonsensitive information about P and can reproduce the experiment on similar hardware and software environments.

An Example. Suppose P is the process of training a deep learning model on a Spark [1] system, and l is a parsed log file produced during the training process. The owner wants to share the log file but is concerned that it may reveal the number of cores of the physical machines used to train the models. In this case, the number of cores is the sensitive attribute X while other information, such as the training algorithm and software configurations, is nonsensitive.

We assume that the adversary knows a set of possible values for the number of cores (e.g. $C = \{1, 2, 4, 8\}$). In addition, the adversary has access to a variety of machines with different number of cores, the same system environment as the user (i.e. the Spark system), and the training algorithm used by the user (program P). The goal of the adversary is to infer the number of cores of the machines. To gather information for the inference, the adversary can generate multiple log files (l) by running the process P on machines with different number of cores. This strong adversary model allows us to provide protection against adversaries with different background knowledge in practice.

3.2 Log Indistinguishability

In this section, we propose a game-based definition to formalize the inference process. Based on this definition, we discuss the requirements for preventing the leakage of the sensitive information.

The Distinguishing Game. Given a process P and a candidate set C, we model the problem of inferring the sensitive attribute as the following distinguishing game between the user and the adversary:

(1) The user picks a value $x \in C$ uniformly at random and generates a log file l.
(2) The user invokes the adversary to obtain a guess $x' = \mathcal{A}(l, C)$.
(3) The attack succeeds if $x' = x$. Otherwise, the attack fails.

In the distinguishing game, the adversary aims to infer the sensitive attribute that is used to produce the log file l. She has access to the candidate set C, which contains a set of potential values for the sensitive attribute. The adversary obtains a guess $x' \in C$ based on some inference strategy $\mathcal{A}$. The attack succeeds only if $x' = x$.

Log Indistinguishability. A program P is γ*-log indistinguishable* on C if, for all $C_{\text{pair}} \subseteq C$ with $|C_{\text{pair}}| = 2$, no adversary can succeed the above distinguishing game with probability greater than $(1 + \gamma)/2$ on C_{pair}.

Log indistinguishability guarantees that no adversary can get a significant advantage over a random guess in inferring the sensitive attribute among any pair of possible values. Therefore, the protection holds even when the adversary is able to eliminate some

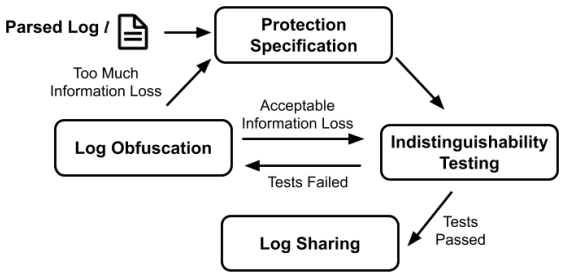


Figure 1: Framework Overview

possible values in C. Specifically, when $\gamma = 0$, no inference strategy outperforms random guessing in winning the distinguishing game, indicating that the log file l leaks no information about the sensitive attribute.

3.3 Framework Overview

Figure 1(a) presents an overview of the log-sharing process, which is inspired by software testing process. In software testing, a set of test modules are designed to identify violations of user-specified requirements. Developers use the test modules to identify bugs in the software. The software is ready to be deployed only if all tests are passed. This process may involve several iterations between implementation and testing. Although software testing cannot guarantee the software to be bug-free, it is the major approach to detect implementation errors and software defects.

Similarly, our testing-based framework aims to provide users an efficient and effective approach to the identification and protection of information leakage in log files. The framework checks an input log file against a protection criterion specified by the user. If the log file fails to meet the criterion, the framework would provide the user with a set of applicable obfuscation methods. The user decides whether the obfuscated log incurs too much information loss and updates the protection specification accordingly. The log can be confidently shared when it passes all the tests. Similar to software testing, indistinguishability testing cannot guarantee that shared log leaks no information. However, it is a practical method to provide confidence that sharing the log has low risk. Moreover, the framework views the log-generating process as a black-box and can therefore be applied to different systems and processes. Below, we briefly introduce each stage in the log-sharing framework:

(1) **Protection Specification.** A user specifies the sensitive attribute X and the candidate set C for a log-generating process P. The goal of the protection is to have P achieve γ-log indistinguishability on C given a small constant γ.
(2) **Indistinguishability Testing (Section 4).** A set of indistinguishability tests are performed to identify violations of the protection criterion.
(3) **Log Obfuscation (Section 5).** The log needs to be obfuscated if it fails any of the indistinguishability tests. We study a range of existing obfuscation methods and propose novel methods to protect the sensitive attribute.
(4) **Log Sharing.** When all the indistinguishability tests are passed, the user can share the log under a much lower risk of exposing the sensitive attribute.

4 INDISTINGUISHABILITY TESTS

In this section, we introduce a set of indistinguishability tests to identify potential violations of γ-indistinguishability. First, we introduce some general principals for designing indistinguishability tests. Then, we propose four types of tests that cover different information in logs. Finally, we introduce methods to interpret and combine the results from different tests.

4.1 A Testing-Based Approach

Challenges in Obtaining Theoretical Guarantee. To obtain the theoretical guarantee of γ-log indistinguishability, one could either (i) prove that all the metrics influenced by the sensitive attribute are removed or (ii) show that the obfuscation methods have effectively hidden all the influence of the sensitive attribute. However, both approaches are challenging in practice.

For the first approach, it is difficult to identify *all* metrics that are influenced by the sensitive attribute. Moreover, there are complex correlations between the capacity of physical machines, software and hardware configurations, and performance metrics. Therefore, obtaining a theoretical proof on the influence of a software or hardware setting is generally impractical. Theoretical analysis of a system usually relies on the assumptions that the data obtained from the system follow a known distribution (e.g. Gaussian distribution) [26]. However, many studies suggest that these assumptions do not hold in practice [18, 25].

For the second approach, most obfuscation methods suffer from the lack of certainty—one cannot prove the effectiveness of commonly used obfuscation mechanisms [33]. Meanwhile, obfuscation methods that do provide certainty for protection, such as differential privacy, often reduces the accuracy in log analysis by adding too much noise.

A Testing-Based Approach. Although obtaining theoretical guarantee is challenging, it is feasible to identify patterns in a log file that violate γ-log indistinguishability. In this section, we take a hypothesis testing approach to understand the risk of inadvertently revealing the sensitive attribute by sharing log files. We design a set of *indistinguishability tests* that provide a user with an empirical understanding on the risk of leaking the sensitive attribute by sharing the log files. The tests do not modify the log files. Based on the test results, a user can decide whether it is necessary to redact or obfuscate any metrics before releasing the logs.

This testing-based approach is similar to running a randomness test for a Pseudo-Random Number Generator (PRNG). The randomness test consists of a set of hypothesis tests that identify non-random patterns in psuedo-random sequences. If the randomness test fails, there is likely to be flaw in the design or implementation of the PRNG. Meanwhile, passing the randomness test only gives a user higher confidence in the quality of a PRNG. There is no guarantee for randomness even if the sequences pass all randomness tests. Similarly, passing the indistinguishability tests do not theoretically guarantee γ-log indistinguishability. However, it gives the user a higher confidence (quantified by γ) that the sensitive attribute is unlikely to be leaked through the released log files.

4.2 Steps of Indistinguishability Testing

The goal of indistinguishability testing is to identify violations of γ-log indistinguishability. Therefore, the null hypothesis ($\mathcal{H}_0$) under test is that P is γ-log indistinguishable on a candidate set C. Associated with this null hypothesis is the alternative hypothesis ($\mathcal{H}_A$) that P is not γ-log indistinguishble. The testing process consists of three steps: (i) generation of test logs, (ii) selection of mapping functions, and (iii) performing hypothesis tests.

Step 1: Generate Test Logs. The first step is to obtain a set of logs to perform the tests on. For each sensitive attribute $x_i \in C$, a set of n parsed log files $\mathcal{L}_i = \left\{ l_i^{(k)} \;\middle|\; k \in \mathbb{Z}^+, k \leq N \right\}$ are generated by running the process for N times.

Step 2: Select Mapping Functions. Due to the high dimensionality of each parsed log file l, directly performing hypothesis tests on l requires an impractically large number of samples. Therefore, a *mapping function* $f : l \mapsto u$ is used to map l to a lower-dimensional vector u on which hypothesis tests can be efficiently performed. For example, the mapping function f_{length} returns the length of a log file (i.e. number of measurements over time). To improve test coverage, one should select a variety of mapping functions that cover different aspects of the log.

Step 3: Perform Hypothesis Tests. Finally, for each pair of $x_{i_1}, x_{i_2} \in C$, the user obtains two groups of outputs:

$$U_{i_1} = \left\{ f\left(l_{i_1}^{(k)}\right) \;\middle|\; l_{i_1}^{(k)} \in \mathcal{L}_{i_1} \right\}, \quad U_{i_2} = \left\{ f\left(l_{i_2}^{(k)}\right) \;\middle|\; l_{i_2}^{(k)} \in \mathcal{L}_{i_2} \right\}. \quad (1)$$

A hypothesis test T is performed on U_{i_1} and U_{i_2} to determine whether the output of the mapping function f is sufficient to distinguish between x_{i_1} and x_{i_2}.

Each indistinguishability test is a combination of a mapping function f and a hypothesis test T. In the following subsections, we propose a test suite consisting of different pairs of (f, T) that cover various aspects of a log file. The test suite can be used similarly to the randomness test suite [34]. Each test returns an independent result on whether there is a risk of information leakage. In Section 4.4, we propose an analytical approach that combines the test results and leads to a conclusion on the risk of revealing the sensitive attribute.

4.3 Designing Indistinguishability Tests

An indistinguishability test consists of two parts: a mapping function f that extracts information from an parsed log file l and a hypothesis test T that checks whether the extracted information leaks the sensitive attribute. In the following subsections, we propose four different types of mapping functions f and associate them with different hypothesis tests T. Specifically, when $\gamma = 0$, we perform kernel tests [16] and χ^2 tests [24] on the outputs of f. When $\gamma > 0$, we connect γ-indistinguishability with ε-differential privacy and perform differential privacy tests [10].

4.3.1 Mapping Functions. There are infinite number of functions that could extract useful information from a parsed log file, and it is impractical to design a "complete" set of mapping functions to cover all possible information leakage. Therefore, to strike a balance between the efficiency and completeness, we propose a set of mapping functions $\mathcal{F}$ that meets two criteria: (i) each mapping function $f \in \mathcal{F}$ returns a low-dimensional numerical/categorical vector; (ii) different mapping functions cover different aspects of l. The first criterion ensures that hypothesis tests can be efficiently performed on the output of f, while the second criterion reduces test redundancy and improves test completeness. This idea is similar to performing a set of randomness tests, where each test checks a different statistical pattern in a random sequence.

Length. The length of a parsed log file l refers to the number of measurements that have been recorded in the log. To extract this information for testing, we define a length mapping function f_{length} that returns the length of an input log file l.

Frequency (Categorical). Since there are few hypothesis tests that support comparison between multi-dimensional categorical vectors, we independently compare each categorical metric in l. A *frequency mapping function* $f_{\text{freq},j,t,w}$ returns the count of each value for the j-th categorical metric in the time window $[t, t + w)$. We use a set of frequency mapping functions to extract the information from a sequence of non-overlapping time windows: $\mathcal{F}_{\text{freq},j,w} = \left\{ f_{\text{freq},j,t,w} \mid t = kw, k \in \mathbb{N}, k < L/w \right\}$, where L is the length of the log file. The output of each mapping function $f \in \mathcal{F}_{\text{freq},j,w}$ is a frequency vector containing the count for each value of the categorical metric. The window size w can be adjusted to balance between testing time and testing strength. A smaller w allows the user to identify minor differences between two sets of logs, but requires more tests to be performed. When $w = 1$, we perform one test on each measurement in l.

Moving Average (Numerical). When analyzing numerical metrics, we combine them into a multi-dimensional time series, and apply time series analysis techniques. The *moving average* technique replaces each element in a time series with the average of surrounding elements to eliminate local variations. Similarly, given a series of numerical metrics, we propose a *moving average mapping function* $f_{\text{avg},t,w}$ that calculates each metric's average value in the time window $[t, t + w)$. We use a set of moving average mapping functions to extract the information from a sequence of non-overlapping time windows: $\mathcal{F}_{\text{avg},w} = \left\{ f_{\text{avg},t,w} \mid t = kw, k \in \mathbb{N}, k < L/w \right\}$. The output of each mapping function $f \in \mathcal{F}_{\text{avg},w}$ is a multi-dimensional numerical vector whose dimension equals to the number of numerical metrics in the log.

Moving Difference (Numerical). Local variations in a time series could also leak sensitive information. A common technique to study the local variation is to calculate the difference between consecutive measurements. Therefore, we propose a *moving difference mapping function* $f_{\text{diff},t}$ that returns the difference between a measurement at time $t + 1$ and a measurement at time t for all the numerical metrics. We use a set of moving difference mapping functions to extract the difference between each consecutive measurements: $\mathcal{F}_{\text{diff}} = \left\{ f_{\text{diff},t} \mid t \in \mathbb{N}, t < L - 1 \right\}$. Similar to the moving average function, the output of each mapping function $f \in \mathcal{F}_{\text{diff}}$ is a multi-dimensional numerical vector whose dimension equals to the number of numerical metrics in the log. When L is large, performing tests on the output of each function in $\mathcal{F}_{\text{diff},j}$ can be time-consuming. Therefore, we pick s mapping functions from $\mathcal{F}_{\text{diff}}$ uniformly at random, and obtain $\mathcal{F}_{\text{diff},s} \subseteq \mathcal{F}_{\text{diff}}$.

4.3.2 Considerations for Choosing Hypothesis Tests. For each mapping function f, we obtain two sets of outputs U_{i_1} and U_{i_2} (E.q. 1) that are associated with sensitive attributes x_{i_1} and x_{i_2} respectively. The next step is to select a two-sample hypothesis test T that takes U_{i_1}, U_{i_2} as inputs and identifies violations of γ-log indistinguishability. The hypothesis test T should meet two criteria:

(1) *Correctness:* If the log-generating process P is γ-log indistinguishable, the probability of rejecting the null hypothesis under significance level α should be no greater than α.
(2) *Power:* Among all known tests that satisfy the correctness criterion, the test with the strongest power should be selected.

The correctness criterion minimizes the type I error—the probability of rejecting log files that satisfy γ-indistinguishability. This criterion guarantees usability of the test. In software testing, if a unit test often fails on correct code, its result will be ignored by developers. Similarly, if a hypothesis test has a large type I error, the test result becomes unaccountable. To guarantee the correctness criterion, the null hypothesis of T ($\mathcal{H}_0$) needs to be a necessary condition for γ-log indistinguishability (i.e. $\mathcal{H}_0$ should always hold when P satisfies γ-log indistinguishability).

The power criterion minimizes the probability of accepting log files that violate γ-indistinguishability. If the process P passes a test with stronger power, the sensitive attribute is less likely to be leaked.

4.3.3 Tests of 0-Log Indistinguishability. When P is 0-log indistinguishable on $\{x_{i_1}, x_{i_2}\}$, the outputs U_{i_1}, U_{i_2} of any mapping function should have the same distribution. Otherwise, the difference between U_{i_1} and U_{i_2} would not allow the adversary to gain advantage in distinguishing between x_{i_1} and x_{i_2}. Therefore, given two samples U_{i_1}, U_{i_2}, the null hypothesis under test is

$$\mathcal{H}_0 \quad : \quad U_{i_1}, U_{i_2} \text{ have the same distribution.}$$

There are several hypothesis tests that can check whether two samples have the same distribution, such as the t-test [40], the KS test [39], χ^2 test [24], and kernel two-sample test [16]. Some of these tests, such as t-test, assume that the two samples come from a known distribution (e.g. normal distribution). Since these assumptions do not always hold in practice [25], we choose among non-parametric tests that do not rely on any assumptions about the underlying distribution of the samples. Specifically, we use χ^2 tests for categorical metrics and kernel two-sample tests for numerical metrics because they are shown to be more powerful than other tests that serve the same purpose [16]. Additionally, since kernel two-sample tests support comparisons between multi-dimensional numerical vectors, the outputs from multiple numerical metrics can be tested together. This approach could identify potential information leakage from the correlation between different metrics.

Table 1 shows the association between different mapping functions and the hypothesis tests to be performed on the outputs of these functions. Based on the types of mapping functions, we name the indistinguishability tests `length test`, `frequency test`, `moving average test`, and `moving difference test`.

4.3.4 DP Test. If the user is willing to tolerate a small amount of information loss when releasing the log, he could adjust the risk of revealing the sensitive attribute by setting the constant γ. However, when $\gamma > 0$, the tests in Section 4.3.3 are no longer applicable because U_{i_1} and U_{i_2} could come from slightly different distributions. Hence, we use DP tests [10] to check γ-log indistinguishability with $\gamma > 0$. Consider a mechanism $\mathcal{M}_{P,f} : x \mapsto u$ consisting of two steps: (i) generate a parsed log file l by running P with sensitive attribute x; (ii) compute the output $u = f(l)$. The following theorem shows the connection between γ-indistinguishability and DP.

$\mathcal{F}$	**Size of $\mathcal{F}$**	**Output Format**	**Test**
Length	1	Integer	Kernel Test
Frequency	$\lceil L/w \rceil$	Count	χ^2 Test
Moving Average	$\lceil L/w \rceil$	Numerical vector	Kernel Test
Moving Difference	s	Numerical vector	Kernel Test

Table 1: Mapping Functions in Indistinguishability Tests.

Theorem 4.1. *If P is γ-indistinguishable on C, for all mapping function f, $\mathcal{M}_{P,f}$ is ε-differentially private on any pair of $x_{i_1}, x_{i_2} \in C$, where $\varepsilon = \log((1+\gamma)/(1-\gamma))$.*

Proof. Let $p_1 = \Pr[\mathcal{M}_{P,f}(x_1) = u]$ and $p_2 = \Pr[\mathcal{M}_{P,f}(x_2) = u]$. Suppose an adversary guesses the sensitive attribute to be x_1 if $p_1 > p_2$, and x_2 otherwise, then his probability of success $p_{\text{win}} = \max(p_1, p_2)/(p_1 + p_2)$. If $\mathcal{M}_{P,f}$ is not ε-differentially private, there exists an output u so that $\max(p_1/p_2, p_2/p_1) > \exp(\varepsilon)$. Since $\varepsilon = \log((1+r)/(1-r))$, we have $p_{\text{win}} > (1+\gamma)/2$. □

Therefore, the indistinguishability test fails if the DP test fails on any pair of U_{i_1}, U_{i_2} generated by the mapping functions in Section 4.3.1. DP tests are applicable to both categorical and numerical data and can be performed on the outputs of all mapping functions.

However, since the DP test is an adapted form of binomial tests [10], it is not as powerful as the tests listed in Section 4.3.3. Consequently, there is a higher risk that the tests could incorrectly accept log files that are not γ-indistinguishable. Adapting more powerful statistical tests for DP and γ-log indistinguishability is nontrivial and retained for future work.

4.4 Interpretation of Test Results

Similar to a randomness test suite [34], an indistinguishability test suite consists of multiple tests that cover different aspects of a log file. Each test has a unique mapping function f that extracts some particular information from the log. Specifically, the mapping functions either (i) belong to different mapping function sets $\mathcal{F}$ (Table 1); or (ii) extract information from different sub-sequences in l (e.g., mapping functions in $\mathcal{F}_{\text{avg},w}$ calculate the moving average in different time windows). Based on these two differences, we introduce two methods to combine multiple test results.

Combining Test Results over Different Sub-Sequences. Suppose $(p_1, p_2, \ldots, p_k)$ is a sequence of p-values returned by performing the same type of indistinguishability test on different sub-sequences of the logs. Under the null hypothesis (i.e., the log-generating process is γ-log indistinguishable), the p-values should be uniformly distributed [7]. We use the Fisher's method [13] to test the uniformity of the p-values.

Combining Test Results over Different Mapping Function Sets. By combining the results of tests performed on different sub-sequences, we obtain one p-value associated with each mapping function

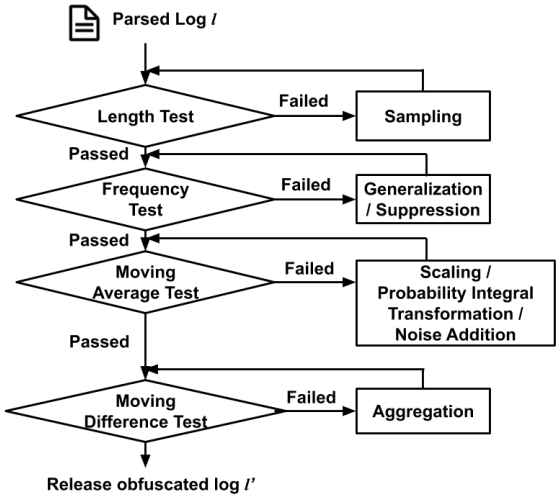


Figure 2: Testing-Based Log Obfuscation Framework

set. Since different mapping function sets focus on different metrics/statistics in the logs, each set represents a unique attack surface for the adversary. For example, if the test associated with the length mapping function fails, it is likely that the adversary could infer the sensitive attribute based on the length of the log. Therefore, the p-value associated with different mapping function sets should be interpreted independently. If any of the p-value is smaller than the significance level α, there is a risk of revealing the sensitive attribute. Additionally, failed tests also indicate the statistics associated with the information leakage. In Section 5, we introduce obfuscation methods to mitigate the risk identified by each test.

5 PROTECTIONS WITH LOG OBFUSCATION

In practice, companies often apply obfuscation techniques to hide sensitive information in log. For example, Reiss et al. [33] proposed log-obfuscation techniques for releasing the Google's cluster traces. However, their approach is limited by the lack of understanding on the protection goal of each obfuscation technique. Therefore, it is unclear whether these techniques are effective in hiding the sensitive information and whether it is necessary to apply them under a different system or a different protection goal.

In this section, we combine the indistinguishability tests proposed in Section 4.3 with different obfuscation techniques. The results of the tests suggest whether a specific obfuscation technique would be helpful in protecting the user-specified sensitive attributes. Additionally, we find out that existing obfuscation techniques are not sufficient to hide all the information identified by our indistinguishability tests and propose two novel obfuscation techniques: probability integral transformation and noise addition. By combining these techniques, we propose an end-to-end framework (Figure 2) to help users identify sources of potential information leakage and mitigate it. Specifically, we use sampling to mitigate leakage identified by length tests, generalization and suppression to mitigate leakage identified by frequency tests, and aggregation to mitigate leakage identified by moving difference tests. Meanwhile, a moving average test could identify leakage from three possible sources: (i) the magnitude of a numerical metric; (ii) the distribution (e.g. variation and skewness) of a numerical metric; (iii) the correlations between different numerical metrics. Therefore, we use three different obfuscation methods to mitigate the leakage associated with each source: (i) sampling, (ii) probability integral transformation; (iii) noise addition.

Sampling. Sampling refers to the process of selecting a subset of measurements in a log file. For example, the user could release a set of measurements uniformly sampled from the whole log file.

Generalization and Suppression. Generalization and suppression are often used to hide sensitive information in categorical attributes [41]. They can be applied to categorical metrics when information leakage is identified by a frequency test.

Scaling. Scaling could mitigate the information leakage from the magnitude of a numerical metric. For example, in the released Google cluster trace dataset, Reiss et al. [33] re-scaled the machine capacity data to guarantee that the maximum observed value is 1.

Probability Integral Transformation. Suppose X is a random variable with cumulative distribution function (cdf) F_X. Probability integral transformation (PIT) converts X to a different random variable Y with cdf F_Y. It relies on the property that $Z = F_X(X)$ follows a uniform distribution. Therefore, the random variable $Y = F_Y^{-1}(F_X(X))$ follows the distribution defined by F_Y.

Suppose S_1, S_2, S_3 are three sets of measurements obtained under sensitive attributes x_1, x_2, x_3 respectively. To perform the obfuscation, one needs to first estimates the empirical cdf F_{mix} of $S_1 \cup S_2 \cup S_3$. Then, for each set S_i ($i \in \{1, 2, 3\}$), by applying PIT on each value in S_i, one obtains obfuscated measurements that follow the new distribution defined by cdf F_{mix}. PIT mitigates the information leakage from the distribution of a numerical metric by ensuring that the measurement taken under different sensitive attributes share the same distribution.

Noise Addition. The key limitation of PIT is that it needs to be *independently* applied to each numerical metrics. Hence, PIT could not prevent information leakage from the correlation between different attributes. For example, suppose two numerical metrics m_1 and m_2 have a positive correlation. This correlation would be retained after PIT. Therefore, if this correlation leaks the sensitive attribute, one needs to add noise to both m_1 and m_2 to hide it. We add Gaussian noise that follows the distribution $\mathcal{N}(0, \sigma^2)$, and determined σ based on the result of the moving average test.

Aggregation. Failing of the moving difference test indicates that the local variations in the numerical metrics might reveal the sensitive attribute. Hence, one could use aggregation techniques to prevent the leakage. For example, instead of revealing all the measurements, one could calculate the average of a measurement over a fixed-length time window. The aggregation technique eliminates local variations, but preserves the trend of the metrics.

6 CASE STUDIES

In this section, we performed case studies on two log datasets: a Spark [1] event log dataset and a hardware performance counter (HPC) dataset collected by Ganju et al. [14]. Similar to Google cluster traces [33, 36], Spark event logs provides a variety of performance counters and workload traces. These traces are extensively used to perform performance analysis and modeling [21, 29] for different purposes such as performance prediction, diagnosis and configuration selection[2, 28, 43]. The HPC dataset contains measurements

Tests	Length Test		Frequency Test		Moving Average Test		Moving Difference Test	
γ	0	0.2	0	0.2	0	0.2	0	0.2
p-values	0.81	0.75	0.89	1.00	0.64	1.00	0.62	1.00
Testing Time (s)	<0.01	0.22	0.03	13.05	5.14	52.32	4.74	66.60

Table 2: Test Correctness and Performance Analysis on Spark Event Logs.

Tests	Length Test			Frequency Test			Moving Average Test			Moving Difference Test		
γ	0	0.2	0.4	0	0.2	0.4	0	0.2	0.4	0	0.2	0.4
No Obfuscation	<0.01	<0.01	<0.01	-	-	-	-	-	-	-	-	-
Sampling	0.99	0.77	0.90	0.93	1.00	1.00	<0.01	<0.01	<0.01	-	-	-
Sampling + Scaling	0.99	0.78	**0.91**	0.93	1.00	**1.00**	<0.01	<0.01	**0.17**	-	-	**0.69**
Sampling + Scaling + PIT	**0.99**	**0.79**	0.93	**0.93**	**1.00**	1.00	**0.27**	**1.00**	1.00	**1.00**	**1.00**	1.00

Table 3: Effectiveness of Different Obfuscation Techniques on Spark Event Logs.

about hardware events on a computer system. These measurements have been demonstrated to be useful in detecting malware [9], side-channel attacks [8], and cryptomining behavior [42]. These two case studies cover different data types and different correlations between the sensitive information and the logged metrics.

We evaluated our framework from four aspects: the correctness of the tests, the effectiveness of obfuscation, the risk under attacks, and the utility of the obfuscated logs. Specifically, our evaluations on the obfuscation techniques focused on the two novel techniques proposed in this paper: probability integral transformation (PIT) and noise addition. We also used sampling and scaling to mitigate the information leakage. Our case studies did not cover generalization, suppression, and aggregation.

6.1 Spark Event Log Dataset

Experimental Setup. Spark is an open-source distributed system for large data analysis. A Spark application is automatically divided into several stages, each consisting of multiple tasks that can be executed in parallel. A Spark event log records information about each individual task in a Spark application. We adapted Spark trace analysis tool [29] to parse the event logs into structured multidimensional sequential vectors, where each data point represents a different task. Each parsed log file contains 15 numerical metrics and 4 categorical metrics [1]. The log-generating process P was a Spark application for training a multi-layer perceptron (MLP) model using the SparkML library. The training data were randomly generated using the `sklearn` library. The training dataset contained 10,000 records with 100 features and 2 classes. We considered hardware information as the sensitive attribute. Specifically, we ran the same process P on two clusters of machines provided by Emulab (emulab.net):

- **Cluster 1:** 10 Dell Poweredge R430 1U servers each with two 2.4 GHz 64-bit 8-Core processors and 64GB RAM;
- **Cluster 2:** 10 Dell PowerEdge 2850s each with a single 3GHz processor and 2GB RAM.

The protection goal was to prevent the adversary from correctly guessing the cluster on which P was running.

On each cluster, we obtained 100 log files by repeatedly running P under the same settings. $\mathcal{L}_1$ denotes the set of logs obtained on Cluster 1, and $\mathcal{L}_2$ denotes the set of logs obtained on Cluster 2. Since parallel tasks of the same job often share similar system metrics (e.g. running time, I/O size), we randomly sampled 1 task per stage when performing the moving average tests and the moving difference tests to prevent test redundancy. The sampled time sequences have a maximum length of 48. We set the window size $w = 1$ for frequency tests and moving average tests. All the tests were performed locally on a laptop with single 2.7 GHz Intel Core i5 processor.

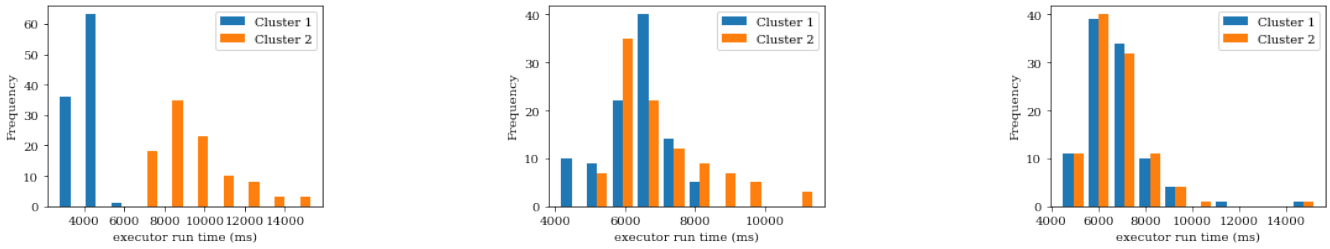
Correctness and Performance. We performed experiments to demonstrate that the indistinguishability tests have low testing overhead, and the tests could stably accept the null hypothesis $\mathcal{H}_0$ (i.e., P is γ-log indistingsuihable) when $\mathcal{H}_0$ holds. An indistingusihability test is correct only if it rejects the null hypothesis with a low probability ($\Pr[p < \alpha \mid \mathcal{H}_0] < \alpha$) when P satisfies γ-log indistinguishability. To check the correctness of the tests, we performed the tests on two groups of logs generated on the same cluster. Specifically, we randomly divided $\mathcal{L}_1$ into two groups G_1 and G_2, each containing 50 parsed log files. We performed indistinguishability tests on G_1 and G_2 with the null hypothesis that G_1 and G_2 are γ-log indistinguishable. We repeated the testing process for 10 times with $\alpha = 0.01$. In all the 10 repetitions, the log files passed the indistinguishability tests for both $\gamma = 0$ and $\gamma = 0.2$. Table 2 presents the average p-values for each test.

Additionally, we evaluated the average testing time for each test (Table 2). The total test time is around 10 seconds when $\gamma = 0$ and around 2 minutes when $\gamma > 0$. When $\gamma = 0$, the tests took less time because χ^2 tests and kernel tests are more efficient than DP tests on multi-dimensional data.

Testing-Based Obfuscation. We performed different obfuscation techniques based on the test results and demonstrated that these techniques could effectively reduce the risk of information leakage. We evaluated the log-sharing framework on the two sets of logs $\mathcal{L}_1$ and $\mathcal{L}_2$ obtained on different clusters. We followed the iterative process shown in Figure 2: when the logs failed an indistinguishability test, we applied obfuscation techniques and repeated the failed tests; when the logs passed a test, we moved to the next test. The logs could be shared after all tests are passed.

We applied three different obfuscation techniques. When the length test failed, we sampled 3 parallel tasks per stage and discarded measurements of remaining tasks in the log file. When the

[1] A complete metric list is available on spark.apache.org/docs/latest/monitoring.html



(a) Original (b) Scaled (c) Scaled + PIT

Figure 3: Analysis on Probability Integral Transformation (PIT).

moving average test failed, we scaled the numerical metrics based on the following strategy: (i) we calculated the medians m_1 and m_2 in $\mathcal{L}_1$ and $\mathcal{L}_2$ for each numerical metric; (ii) we scaled each numerical metric in log $\mathcal{L}_i$ by multiplying it with a constant $c_i = \frac{m_1+m_2}{2m_i}$. This scaling strategy ensures that numerical metrics in $\mathcal{L}_1$ and $\mathcal{L}_2$ share the same median ($\frac{m_1+m_2}{2}$) while preserving the relative magnitudes of these metrics. If the scaled logs still failed the moving average tests, we further applied PIT to ensure that the metrics in $\mathcal{L}_1$ and $\mathcal{L}_2$ share the same distribution (Section 5).

We performed indistinguishability tests with $\alpha = 0.01$ under three different settings: (i) $\gamma = 0$, (ii) $\gamma = 0.2$, and (iii) $\gamma = 0.4$. Based on the definition of γ-log indistinguishability, the settings could be translated into three levels of protection objectives against an adversary trying to infer the sensitive attribute: (i) the adversary's performance should be equivalent to random guessing; (ii) the attack accuracy should be lower than 0.6; (iii) the attack accuracy should be lower than 0.8.

Table 3 presents the p-value of each test under different γ and obfuscation techniques. The values in bold indicate that the obfuscated logs passed all tests (i.e., $p > 0.01$) and could be shared. When $\gamma = 0$ and $\gamma = 0.2$, sampling, scaling, and PIT were required to achieve the protection criteria. Meanwhile, when $\gamma = 0.6$, PIT was not needed to ensure the protection criterion.

When no obfuscation was applied, logs generated on Cluster 1 and Cluster 2 had different length. Since machines in Cluster 1 have more processors compared to machines in Cluster 2, they could support more parallel tasks, which resulted in a larger log file. This leakage was mitigated by sampling a subset of tasks per stage.

However, after sampling, the logs still leaked sensitive information through the magnitude of numerical metrics. Since the same job was divided into more parallel tasks on Cluster 1, tasks running on Cluster 1 had shorter duration and smaller I/O size. An adversary could use this information to infer the cluster on which the logs were generated. Scaling was not sufficient to prevent the leakage because the task metrics on Cluster 1 and Cluster 2 were different not only in their magnitudes but also in the variation and skewness of their distributions.

Figure 3 shows the distributions of `executor run time (ms)` of Task 0 in Stage 0. Prior to the obfuscation (Figure 3a), the distributions were different in two aspects: (i) tasks on Cluster 1 had shorter runtime than tasks on Cluster 2; (ii) runtime on Cluster 2 had a right-skewed distribution, indicating that there were more stragglers (i.e., tasks with runtime larger than 1.5× the median runtime). The scaled metrics (Figure 3b) shared similar magnitudes,

	Attack Acc.			# of Correct Ans.		
	MLP	NN	Length	<10	10-15	16
No Obfuscation	1.00	1.00	1.00	0	0	100%
Sampling	0.95	1.00	0.55	0	27%	73%
Sampling + Scaling	0.58	0.65	0.55	0	27.5%	72.5%
Sampling + Scaling + PIT	0.43	0.55	0.55	0.5%	30.5%	69%

Table 4: Attack and Utility Analysis on Spark Logs.

but did not eliminate the stragglers on Cluster 2. Therefore, an adversary could infer the sensitive information by identifying the stragglers in the log file. After applying PIT (Figure 3c), the number of stragglers increased on Cluster 1 and decreased on Cluster 2. Since the two sets of obfuscated logs shared the same distribution, an adversary could no longer infer the cluster on which the logs were generated.

Risk under Different Attacks. We designed three different attacks to verify the test results. We showed that an adversary could correctly infer the sensitive attribute when the tests failed. Moreover, when the logs were obfuscated and passed the tests, the attack accuracy dropped to around 0.5.

Specifically, we modeled an attack as a classification problem with the parsed log files as input features and the cluster information as class labels. We randomly divided $\mathcal{L}_1 \cup \mathcal{L}_2$ into equal-sized datasets $\mathcal{L}_{train}$ and $\mathcal{L}_{test}$ each containing 50 parsed logs from Cluster 1 and 50 parsed logs from Cluster 2. We assumed that an adversary had access to the labels (i.e. cluster information) for logs in $\mathcal{L}_{train}$ and wanted to predict the labels of logs in $\mathcal{L}_{test}$. In practice, an adversary could obtain $\mathcal{L}_{train}$ and their labels by running P on her own machines, and $\mathcal{L}_{test}$ represents the logs shared by the users.

We trained three different attack classifiers: (i) a multi-layer perceptron (MLP) model with 100 hidden units; (ii) a nearest neighbor (NN) classifier; and (iii) a length classifier that predicts the cluster label based on the length of a parsed log file. Table 4 shows the attack accuracy on $\mathcal{L}_{test}$ of the three attacks. Without obfuscation, all attacks achieved high accuracy. Sampling could only protect against the length attack, and scaling could reduce the attack accuracy to below 0.7. This result complies with our previous test results for $\gamma = 0.4$. After PIT, all attack accuracy dropped to around 0.5, indicating that the obfuscated logs were likely to satisfy 0-indistinguishability.

Utility of Obfuscated Logs. We demonstrated that the obfuscated logs were still useful in helping users identify bottlenecks of the system. We performed the analysis proposed by Ousterhout et al [29] to study the utility of obfuscated logs. The goal of the analysis is to identify performance bottlenecks in a system. It estimates how long the application is blocked on a certain aspect of the system (e.g. network, computation, I/O). To get a quantitative understanding of the analysis results, we designed 16 questions that could be answered based on the analysis results:

(1) What is the bottleneck of the system?
(2-4) Is blocked time on computation greater than 20%/50%/80%?
(5-7) Is blocked time on disk greater than 20%/50%/80%?
(8-10) Is blocked time on GC greater than 20%/50%/80%?
(11-13) Is blocked time on network greater than 20%/50%/80%?
(13-15) Is blocked time on stragglers (i.e., tasks with runtime larger than 1.5× the median runtime) greater than 20%/50%/80%?
(16) What is the number of stragglers?

We performed the analysis for each $l \in \mathcal{L}_1 \cup \mathcal{L}_2$, and compared the answers obtained from original logs and obfuscated logs. Table 4 presents the number of questions that could be correctly answered on the obfuscated logs. After obfuscation, the analysis process could still correctly answer all the questions on around 70% of the obfuscated log files. Among the 16 questions, the questions about stragglers (Q13-Q16) were most likely to be influenced by the obfuscation techniques. Since tasks on Cluster 2 had more stragglers compared to tasks on Cluster 1, giving out accurate information about the stragglers would inevitably leak information about the clusters. Therefore, there is a trade-off between hiding the sensitive information and retaining useful information in obfuscated logs. Our testing-based framework provides a better understanding on this trade-off by helping users intuitively understand the effectiveness of different obfuscation techniques.

6.2 Hardware Performance Counter Dataset

Experimental Setup. In this case study, we studied the risk associated with sharing HPC datasets. For example, suppose a small company wants to detect whether its employees have been using the company resources for covert cryptomining. The company could outsource this detection to a third-party service by sharing its HPCs. However, the company might be concerned about leaking information about the security vulnerabilities of its machines. In this case, the sensitive attribute to be protected is whether the machines are patched against Spectre [19] and Meltdown [22] attacks.

Our dataset [14] includes 22 different hardware counters (i.e., numerical metrics) for a variety of cryptomining and non-mining applications. Each record in the dataset contains 3-5 measurements taken at an interval of 2 seconds. The dataset contains two parts: the unpatched subset was generated by running applications on a machine vulnerable to Spectre and Meltdown attacks; the patched subset was generated by running the same applications on the same machine after the patches have been installed. We generated $\mathcal{L}_{\text{p}}$ and $\mathcal{L}_{\text{up}}$ by randomly sampling from the patched and unpatched subsets respectively. $\mathcal{L}_{\text{p}}$ and $\mathcal{L}_{\text{up}}$ both contain 50 records from cryptomining applications and 50 records from non-mining applications. The sampled time sequences have a maximum length of 48.

Tests	**Length**		**Moving Average**		**Moving Difference**	
γ	0	0.2	0	0.2	0	0.2
***p*-values**	1.00	1.00	0.71	0.99	0.67	0.98
Time (s)	<0.01	0.04	0.66	15.01	0.53	12.31

Table 5: Correctness and Performance on HPCs.

Tests	**Length**		**Moving Average**		**Moving Difference**	
γ	0	0.2	0	0.2	0	0.2
No Obfuscation	0.89	1.00	<0.01	<0.01	-	-
Scaling	0.89	1.00	<0.01	<0.01	-	-
PIT	0.89	**1.00**	<0.01	**0.99**	-	**0.72**
Scaling + PIT	0.89	1.00	<0.01	0.99	-	0.81
PIT + Noise	**0.89**	1.00	**0.02**	0.92	**0.48**	0.86

Table 6: Effectiveness of Different Obfuscation on HPCs

Correctness and Performance. We demonstrated that the tests were correct and they incurred little testing overhead on the HPC dataset. We randomly divided $\mathcal{L}_{\text{p}}$ into two groups G_1 and G_2. We performed indistinguishability tests on G_1 and G_2 with the null hypothesis that they are γ-log indistinguishable with $\gamma = 0$ and $\gamma = 0.2$. We repeated the testing process for 10 times with $\alpha = 0.01$. In all the 10 repetitions, the indistinguishability tests were passed for both $\gamma = 0$ and $\gamma = 0.2$. Table 5 presents the average p-values and running time for each test.

Testing-Based Obfuscation. We showed that the obfuscation techniques that were effective on the Spark event logs could not protect the HPC dataset, but the sensitive information could be effectively hidden by the noise addition technique we proposed.

We applied three obfuscation techniques to protect the HPC dataset: scaling, PIT, and noise addition. Table 6 shows the testing results after each obfuscation technique was applied. We performed indistinguishability tests with $\alpha = 0.01$. The values in bold indicate that the obfuscated logs passed all tests and could be shared. Unlike the Spark event log dataset, the HPC dataset could not be protected by scaling because the main source of information leakage is the distribution, rather than magnitude of the performance counters. For example, compared to unpatched machines, patched machines have a larger variation in the number of executed instructions per second. Moreover, the sensitive attribute in the HPC dataset could be inferred through the correlations between different performance counters, so PIT is not sufficient to protect the information leakage.

To hide the correlations that could leak the sensitive attribute, we added Gaussian noise $\mathcal{N}(0, \sigma^2)$ to each measurements in the HPC dataset. To ensure that the noise we added have relatively same magnitude as the original value, we normalized the metrics prior to noise addition and scaled them back after noise was added. We gradually increased σ until the moving average test was passed. With $\sigma = 0.09$, the moving average test was passed with $p = 0.02$. To reduce false negatives (i.e., incorrectly accept the null hypothesis), we repeated the testing process for 10 times and ensured that the test was passed in all the repetitions.

Risk and Utility of Obfuscated Logs. We showed that the noise addition technique was effective in protecting against the attacks and incurred little utility loss on the HPC dataset.

	MLP Attack	NN Attack	Utility
No Obfuscation	0.60	0.80	0.96
Scaling	0.95	0.93	0.97
Scaling + PIT	0.60	0.58	0.91
PIT + Noise	0.47	0.57	0.90

Table 7: Attack and Utility Analysis on Obfuscated HPCs.

Using the techniques in Section 5, we trained two attack classifiers to predict whether the machines were patched: (i) a multi-layer perceptron (MLP) model with 100 hidden units and (ii) a nearest neighbor (NN) classifier. We did not evaluate the length attack because all the records in the HPC datasets have similar length.

We evaluated the utility of the obfuscated dataset using an MLP model with 100 hidden units. The model was trained to predict whether a record was generated by a cryptomining or non-mining application. Table 7 presents the attack accuracy and utility under different obfuscation techniques. The attack accuracy conforms with the test results presented in Table 6, the obfuscation techniques reduced the maximum attack accuracy from 80% to 57% but still preserved a 90% accuracy in classifying mining and non-mining applications.

Analysis on Noise Addition. We showed that PIT could effectively reduce the amount of noise required for protecting the sensitive information. Noise addition could mitigate any information leakage if the standard deviation of the noise (σ) is large enough. However, adding noise with large σ would incur huge utility loss on the logs. To minimize the utility loss, we applied other obfuscation techniques such as scaling and PIT prior to noise addition. Table 8 presents the effectiveness of these techniques in reducing the amount of required noise to pass the moving average test. By applying PIT prior to noise addition, we increased the utility accuracy by around 30%.

	No Obfuscation	Scaling	PIT
$\sigma_{\min}$	0.86	0.89	**0.09**
Utility	0.62	0.54	**0.90**

Table 8: Obfuscation Prior to Noise Addition.

7 RELATED WORK

Differential Privacy. Recently, differential privacy has been extended to protect against side channel attacks. Xu et al. [50] applied differential privacy to prevent information leakage through timing and message size, and Xiao et al. [47] used differential privacy to mitigate storage side channels in `procfs`. However neither of these works discussed privacy in log-sharing. This scenario is different from side channel attacks in two aspects. First, our framework does not require modifications of the system and does not incur any performance overhead on the system. Instead, users could achieve the protection goal by modifying metrics in the shared log files. Second, our framework is designed for general-purpose log sharing. Therefore, we do not make any assumption on the logged metrics or analysis to be performed on the logs.

Log Anonymization. Prior studies have proposed various methods to hide user-identifiable information in logs. These studies can be classified into two categories: log anonymization and encryption. Log anonymization refers to the process of removing or redacting user identifiers in system logs. For example, there has been extensive research on IP anonymization [4, 5, 23, 31, 48, 49], ranging from truncating all IP addresses to prefix-preserving pseudonymization [48, 49]. Similarly, studies on timestamp anonymization has shown that it's possible to convert timestamps into relevant timestamps, which hide the exact time of the events but preserve the orders [32, 45]. On the other hand, some research has studied the use of encryption mechanisms in log privacy. Encryption methods, such as searchable encryption, can hide sensitive information in logs while allowing the administrator to do certain analysis [30, 35, 44, 46]. However, log anonymization focuses on removing user identifiers and is not enough to prevent the leakage of all the sensitive information. In this work, we focus on information leakage from indirect evidence such as performance metrics.

Randomness Test. Randomness tests are statistical tests that compare and evaluate a sequence to a truly random sequence. Rukhin et al. [34] have proposed a randomness test suite consisting of 15 different random tests, each of which covering different randomness statistics. The design of our indistinguishability test suite shares a similar structure with the randomness test suite. However, to identify information in logs, we performed a different set of statistical tests. Additionally, we designed obfuscation methods that help users to efficiently mitigate the identified risks.

Data Obfuscation. Data obfuscation is a mechanism to protect privacy by adding misleading, false, or ambiguous information [27, 37]. For example, Sweeney [41] proposed the use of generalization and suppression to hide identifying information in a dataset containing person-specific records. Bakken et al [3] designed a set of obfuscation primitives and proposed properties to quantify the usefulness and privacy of the techniques. Some data obfuscation techniques such as generalization and suppression could be used to obfuscate logs. In this work, we combineed these techniques with the testing-based log obfuscation framework to protect sensitive information in logs.

Cryptographic Indistinguishability Obfuscation. Cryptographic indistinguishability obfuscation is a cryptographic primitive that provides a formal notion of program obfuscation. It requires that the obfuscation of two equivalent circuits C_0 and C_1 should be computationally indistinguishable [15]. This property can be guaranteed by algebraic hardness assumptions [15] or the security of other cryptographic primitives such as public-key functional encryption [6]. Similar to other program obfuscation techniques, indistinguishability obfuscation methods are designed to hide information of a program, and are not applicable to the protection of sensitive information in logs.

8 CONCLUSION

We have proposed a test-based framework to identify and mitigate the risk of information leakage in general-purpose log sharing. Our framework contains a set of statistical tests to identify violations of the log indistinguishability property and a variety of obfuscation methods such as probability integral transformation and noise addition. We tested our framework on Spark event logs and logs generated by a hardware performance counter. The framework

effectively identified risks in information leakage and mitigated the risks with two obfuscation techniques.

This work was supported in part by NSF CNS 13-30491 and NSF CNS 14-08944. The views expressed are those of the authors only.

REFERENCES

[1] Apache spark. https://spark.apache.org/.

[2] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 469–482, 2017.

[3] D. E. Bakken, R. Rarameswaran, D. M. Blough, A. A. Franz, and T. J. Palmer. Data obfuscation: Anonymity and desensitization of usable data sets. *IEEE Security & Privacy*, 2(6):34–41, 2004.

[4] J. Biskup and U. Flegel. Transaction-based pseudonyms in audit data for privacy respecting intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 28–48. Springer, 2000.

[5] J. Biskup and U. Flegel. On pseudonymization of audit data for intrusion detection. In *Designing Privacy Enhancing Technologies*, pages 161–180. Springer, 2001.

[6] N. Bitansky and V. Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 171–190. IEEE, 2015.

[7] M. Bland. Do baseline p-values follow a uniform distribution in randomised trials? *PloS one*, 8(10):e76010, 2013.

[8] M. Chiappetta, E. Savas, and C. Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing*, 49:1162–1174, 2016.

[9] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, 2013.

[10] Z. Ding, Y. Wang, G. Wang, D. Zhang, and D. Kifer. Detecting violations of differential privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 475–489. ACM, 2018.

[11] M. Du and F. Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[12] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[13] R. A. Fisher. *Statistical methods for research workers*. Genesis Publishing Pvt Ltd, 2006.

[14] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–633. ACM, 2018.

[15] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 40–49. IEEE, 2013.

[16] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[17] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(6):931–944, 2018.

[18] T. Kalibera, L. Bulej, and P. Tuma. Benchmark precision and random initial state. In *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005)*, pages 484–490, 2005.

[19] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, May 2019.

[20] N. Li, W. Qardaji, D. Su, Y. Wu, and W. Yang. Membership privacy: a unifying framework for privacy definitions. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 889–900. ACM, 2013.

[21] D. Lion, A. Chiu, H. Sun, X. Zhuang, N. Grcevski, and D. Yuan. Don't get caught in the cold, warm-up your {JVM}: Understand and eliminate {JVM} warm-up overhead in data-parallel systems. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 383–400, 2016.

[22] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 973–990, 2018.

[23] E. Lundin and E. Jonsson. Privacy vs. intrusion detection analysis. In *Recent Advances in Intrusion Detection*, 1999.

[24] N. Mantel. Chi-square tests with one degree of freedom; extensions of the mantel-haenszel procedure. *Journal of the American Statistical Association*, 58(303):690–700, 1963.

[25] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci. Taming performance variability. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 409–425, 2018.

[26] C. E. McCulloch and J. M. Neuhaus. Generalized linear mixed models. *Encyclopedia of biostatistics*, 4, 2005.

[27] H. Nissenbaum and F. Brunton. Vernacular resistance to data collection and analysis: A political theory. *First Monday*, 16(5), 2011.

[28] K. Ousterhout, C. Canel, S. Ratnasamy, and S. Shenker. Monotasks: Architecting for performance clarity in data analytics frameworks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 184–200. ACM, 2017.

[29] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun. Making sense of performance in data analytics frameworks. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 293–307, 2015.

[30] T. Pulls, R. Peeters, and K. Wouters. Distributed privacy-preserving transparency logging. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 83–94. ACM, 2013.

[31] C. Rath. Usable privacy-aware logging for unstructured log entries. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 272–277. IEEE, 2016.

[32] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, Nov. 2011. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[33] C. Reiss, J. Wilkes, and J. L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *2012 IEEE Network Operations and Management Symposium*, pages 1279–1286. IEEE, 2012.

[34] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-Allen and Hamilton Inc Mclean Va, 2001.

[35] B. Schneier and J. Kelsey. Cryptographic support for secure logs on untrusted machines. In *USENIX Security Symposium*, volume 98, pages 53–62, 1998.

[36] S. Sebastio, K. S. Trivedi, and J. Alonso. Characterizing machines lifecycle in google data centers. *Performance Evaluation*, 126:39 – 63, 2018.

[37] R. Shokri. Privacy games: Optimal user-centric data obfuscation. *Proceedings on Privacy Enhancing Technologies*, 2015(2):299–315, 2015.

[38] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Schölkopf, and G. R. Lanckriet. Hilbert space embeddings and metrics on probability measures. *Journal of Machine Learning Research*, 11(Apr):1517–1561, 2010.

[39] M. A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American statistical Association*, 69(347):730–737, 1974.

[40] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[41] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):571–588, 2002.

[42] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. Gunter, F. Zaffar, M. Caesar, and N. Borisov. Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 287–310. Springer, 2017.

[43] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: efficient performance prediction for large-scale advanced analytics. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 363–378, 2016.

[44] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *NDSS*, volume 4, pages 5–6, 2004.

[45] J. Wilkes. More Google cluster data. Google research blog, Nov. 2011. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[46] K. Wouters, K. Simoens, D. Lathouwers, and B. Preneel. Secure and privacy-friendly logging for egovernment services. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1091–1096. IEEE, 2008.

[47] Q. Xiao, M. K. Reiter, and Y. Zhang. Mitigating storage side channels using statistical privacy mechanisms. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1582–1594. ACM, 2015.

[48] J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the design and performance of prefix-preserving ip traffic trace anonymization. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 263–266. ACM, 2001.

[49] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 280–289. IEEE, 2002.

[50] M. Xu, A. Papadimitriou, A. Feldman, and A. Haeberlen. Using differential privacy to efficiently mitigate side channels in distributed analytics. In *Proceedings of the 11th European Workshop on Systems Security*, page 4. ACM, 2018.

[51] Y. Yang, Z. Zhang, G. Miklau, M. Winslett, and X. Xiao. Differential privacy in data publication and analysis. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 601–606. ACM, 2012.