Securing Data Analytics on SGX With Randomization

Swarup Chandra, Vishal Karande, Zhiqiang Lin, Latifur Khan, Murat Kantarcioglu, and Bhavani Thuraisingham

University of Texas at Dallas, Richardson TX, USA,
{swarup.chandra, vishal.karande, zhiqiang.lin, lkhan, muratk, bhavani.thuraisingham}@utdallas.edu,

Abstract

Protection of data privacy and prevention of unwarranted information disclosure is an enduring challenge in cloud computing when data analytics is performed on an untrusted third-party resource. Recent advances in trusted processor technology, such as Intel SGX, have rejuvenated the efforts of performing data analytics on a shared platform where data security and trustworthiness of computations are ensured by the hardware. However, a powerful adversary may still be able to infer private information in this setting from side channels such as cache access, CPU usage and other timing channels, thereby threatening data and user privacy. Though studies have proposed techniques to hide such information leaks through carefully designed data-independent access paths, such techniques can be prohibitively slow on models with large number of parameters, especially when employed in a real-time analytics application. In this paper, we introduce a defense strategy that can achieve higher computational efficiency with a small trade-off in privacy protection. In particular, we study a strategy that adds noise to traces of memory access observed by an adversary, with the use of dummy data instances. We quantitatively measure privacy guarantee, and empirically demonstrate the effectiveness and limitation of this randomization strategy, using classification and clustering algorithms. Our results show significant reduction in execution time overhead on real-world data sets, when compared to a defense strategy using only data-oblivious mechanisms.

Keywords: Data Privacy, Analytics, Intel SGX, Randomization

1 Introduction

When computation involving data with sensitive information is outsourced to an untrusted third-party resource, data privacy and security is a matter of grave concern to the data-owner. For example, third-party services offering state-of-the-art predictive analytics platform may be used on data containing private information such as health-care records. An adversary in this environment may control the third-party resource for obtaining records of a specific user, or identifying sensitive patterns in data. Typically, data is protected from such external

adversaries using cryptographically secure encryption schemes. However, direct computation on encrypted data, using techniques such as fully-homomorphic encryption schemes [13], can be inefficient for many practical purposes [21], including data analytics - the focus of this paper.

Recent advances in hardware-based technology such as Intel SGX offers cryptographically secure execution environment, called an *Enclave*, that isolates code and data from untrusted regions within a device. It is natural to leverage the confidentiality and trustworthiness provided by this mechanism, supported by an untrusted third-party server, to efficiently perform large-scale analytics over sensitive data which is decrypted within a secure region. An adversary controlling this server will neither have access to decrypted data, nor will be able to modify computation involving it.

Unfortunately, studies have discovered presence of side-channels that may leak undesirable information from within an enclave. By observing resource access and timing, an adversary can design an attack to derive sensitive information from computation at runtime [14,34]. Nevertheless, mechanisms to eliminate such information leak typically relies on the software developer to hide access patterns with other *non-essential* or dummy resource accesses. These include balanced execution [31] and data-oblivious execution [26]. From the adversarial point of view, these mechanisms add noise to patterns emerging from *essential* computation of a naive implementation. Although using such defenses curb information leak from an SGX enclave and guarantee data privacy, they add significant computational overhead on certain applications in data analytics; in settings involving a large number of parameters, and requiring real-time response [23].

In this paper, we discuss a novel defense mechanism that can achieve lower computational overhead with a trade-off on privacy guarantee, when performing data analytics within an SGX enclave running on a third-party server. In particular, we focus on two classical problems in data analytics, i.e., data classification and clustering. Here, a statistical model is used to predict class labels of given data instances (in classification) or associate them to clusters (in clustering). We generate new *dummy* data instances and interleave them with *user-given* data instances before evaluation. Our proposed defense strategy leverages equivalence in resource access patterns observed by an adversary during evaluation of user-given and dummy data instances. This introduces uncertainty in observed side-channel information in a stochastic manner.

In short, we make the following contributions in this paper.

- We present a defense strategy against side-channel attacks on Intel SGX by randomizing information revealed to the attacker, and asymptotically guaranteeing data privacy.
- We illustrate its application on popular data analytics including decision tree and Naive Bayes classification, and k-means clustering techniques.
- We study the effect of privacy in terms of proportion of dummy data instances employed with respect to user-given data instances, and empirically demonstrate the effectiveness of our defense strategy.

The rest of the paper is organized as follows. We first provide relevant background on Intel SGX and data analytics in §2. We detail the threat model and our defense strategy in §3, and describe relevant implementation techniques in §4. We quantify privacy guarantee of the proposed strategy with respect to the number of dummy data instances in §5, and then present empirical estimates of computational overhead using real-world datasets. We finally discuss related studies in §6, and conclude in §7.

2 Background

2.1 Intel SGX

Intel Software Guard Extensions (SGX) [2] is a set of additional processor instructions to the x86 family, with hardware support to create secure memory regions within existing address space. Such an isolated container is called an *Enclave*, while rest of the address space is untrusted. Data within these memory regions can only be accessed by code running within the enclave. This access control is enforced by the hardware, using attestation and cryptographically secure keys [11] with a trusted processor. The new SGX instructions are used to load and initialize an enclave, as well as enter and exit the protected region. From a developer's perspective, an enclave is entered by calling trusted `ecalls` (enclave calls) from the untrusted application space. The enclave can invoke untrusted code in its host application by calling `ocalls` (outside calls) to exit the enclave. Data from the enclave is always encrypted when it is in memory, but there are cases in which the content should be securely saved outside the enclave. The process of exporting the secrets from an enclave is known as *Sealing*. The encrypted sealed data can only be decrypted by the enclave. Every SGX-enabled processor contains a secret hardware key from which other platform keys are derived. A remote party can verify that a specific enclave is running on SGX hardware by having the enclave perform remote attestation.

Attacks While performing computations within the enclave, an adversary controlling the host OS may infer sensitive and confidential information from side-channels [27]. Assuming the application executed within an enclave is benign, i.e., it does not actively leak information, the attacker may observe input-dependent patterns in data access and execution timing for inferring sensitive information. This is called as *cache-timing attack* [14]. Since OS is allowed to have full control over the page table of an SGX enclave execution, the attacker controlling the OS may know page access patterns. This eliminates noise in side-channels, and is called as *Controlled-channel attack* [34].

Defenses The burden of ensuring efficiency, data privacy and confidentiality lies with the application developer who verifies platform authenticity, and performs guarded memory and I/O access. Therefore, studies have proposed various mechanisms including balanced execution [31] and data-oblivious computa-

tions [26]. In balanced execution, each branch of a conditional statement is forcefully executed by creating dummy operations of data and resource access [27]. Whereas a data-oblivious solution has its control-flow independent of its input data. As mentioned in [26], efficient ORAM techniques [33] cannot be employed for data analytics since it does not hide input-dependent access paths, and is not ideal for applications making large number of memory accesses. However, data-independent access techniques can be used to defend against page-level and cache-level attacks. In our paper, we discuss a solution that significantly reduces sensitive information in side-channels by creating and utilizing dummy data along with the original user-given data during computation.

2.2 Machine Learning

Machine learning is a set of algorithms used to learn and predict patterns in data. With applications such as image recognition, video analytics [3] and text comprehension [15], this growing field in computer science has attracted large attention from both industry and academia. In general, a data instance is a d-dimensional vector whose elements represent characteristic features. A set of such data instances is called a *dataset*. The goal of learning is to identify characteristic patterns in a dataset by training a statistical model, which is later used to evaluate data instances in the future by generalization [9]. In our study, we apply the proposed defense strategy on classification models including decision tree and Naive Bayes, where the problem is to predict class label of a given data instance. The classifier parameters are learned using a disjoint dataset with known class labels. Furthermore, we also demonstrate the defense strategy over k-means clustering algorithm, where the problem is to group similar instances in the dataset. In both these problems, the attacker is interested not only in obtaining input-dependent patterns from side-channel information, but also model parameters and structure that are confidential.

3 Secure Data Analytics

3.1 Threat Model

Analytics on data containing sensitive information is performed on a third-party untrusted server with Intel SGX support. While data-owners have no control over this server, they may establish a cryptographically secure connection to an enclave in the server. Similar to [19], we assume that an attacker controls the untrusted server, and has the ability to interrupt the enclave as desired, by modifying the OS and SGX SDK, to obtain side-channel information from page or cache accesses, page faults, and log files. Nonetheless, code and data within the enclave cannot be modified, except by the data-owner.

The primary goal in an attack is to obtain sensitive information leaked through side-channels from a benign machine learning application running within the SGX enclave. Sensitive information may include model parameters, feature

Symbols	Description
d	# Features
C	# Class Labels
x	Data Instance
y	Class Label
n	Dataset Size
k	# Clusters
T	Set of Clusters
L	# Dummy Data

Table 1: List of symbols.

Model	Parameters	
	Public	Confidential
Decision Tree	n, d, C	$x, y,$ `Tree`
Naive Bayes	n, d, C	$x, y, P(y\|x), P(y)$
K-Means	n, d, C, k, I	x, y, T

Table 2: List of public and confidential parameters. Here, `Tree` indicates model structure, and P indicates probability function.

values of input data, and data distribution statistics. For example, structure of a decision tree (denoted by `Tree`) may be revealed if nodes in the tree are present on different pages, while the attacker tracks the order of execution during evaluation. Similarly, proportion of each cluster (denoted by T) in the k-means clustering algorithm may reveal sensitive data patterns. We term this set of sensitive attributes as *confidential*. A defense mechanism aims to prevent the attacker from inferring confidential attributes through side-channel information. Nevertheless, each learning algorithm has parameters which are data invariant. For example, height of a decision tree (H), number of features in each data instance (d), domain and range of feature values (f), number of class labels (C), number of clusters in k-means clustering (k), and number of iterations for learning (I), remains constant for a given dataset. These parameters can be easily inferred from analyzing algorithmic execution. We assume that the code for each algorithm is publicly available, along with its data invariant parameters. Table 2 lists the associated confidential and public parameters for each algorithm considered, with Table 1 listing the frequently used symbols in this paper.

3.2 Overview

Figure 1 illustrates the overall defense methodology proposed in this paper. An user provides cryptographically secure encrypted data (containing sensitive information) to a third-party untrusted server, along with a pre-trained model. An enclave is established, and the pre-trained model initialized. By requesting a set of data instances into the enclave from application memory through an `ocall`, we decrypt these instances and empirically evaluate the domain and range statistics of each feature. Since we desire that computation involving dummy data instances produce access patterns similar to that of user-given data instances, we generate d feature values uniformly at random within its empirical range to create a dummy instance. After generating L such instances, we shuffle them with user-given data instances in a data-oblivious manner and evaluate each instance in the shuffled dataset sequentially using the pre-trained model that is fully encapsulated within the enclave. By obliviously ignoring results associated with dummy data instances, we obtain the results for user-given data instances.

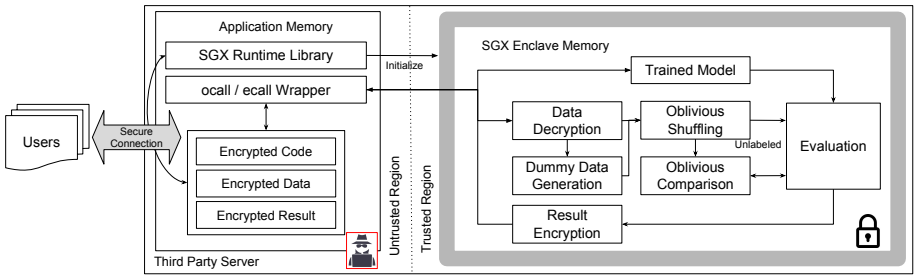


Fig. 1: Overview of Data Analytics on SGX using randomization.

We then encrypt these results in a cryptographically secure manner, and save it in the untrusted application memory via an `ocall`. Here, data-oblivious shuffling of dummy and user-given instances is crucial since it introduces uncertainty in access patterns observed from side-channels by the attacker.

Crux of the above solution is in the way we generate dummy data instances, and use data-oblivious mechanisms for shuffling and ignoring results of computation associated with dummy data instances. If we only employ the shuffled (contaminated) dataset for evaluation in a naive implementation of a data analytics algorithm, i.e., by ignoring results from dummy instances, it may not be possible to conceal all sensitive model parameters and data patterns. Each learning algorithm has an inductive bias, different from one another, which prevents universal application of a naive strategy by itself. For example, the inductive bias of a decision tree is that data can be divided in the form of a tree structure. Whereas, the bias in k-means clustering assumes that instances having similar properties are closer to each other than those with dissimilar properties. In both these cases, the structural representation of data is different, and is input-dependent. We address this challenge by utilizing dummy data instances to conceal model structure and parameters as well. This indicates that computation involving dummy data instances need to be tracked, but in a data-oblivious manner so that uncertainty in resource access trace observed by the attacker is preserved. We first introduce the primitives of our defense strategy, i.e., dummy data generation and data-oblivious comparison, in §3.3, and describe data analytics algorithms that utilize them for defense, in §3.4.

3.3 Primitives

Dummy Data Generation. Algorithm 1 illustrates our dummy data generation process. Using public parameters of user-given dataset D, we choose a random number uniformly within the range of each feature (i.e., values between `MIN` and `MAX`) in D. This choice limits the bias of dummy data instances, and prevents them from having distinguishing characteristics compared to user-given data instances. If not, an attacker may be able to identify such characteristics and discard access traces associated with dummy data instances, thereby defeat-

Algorithm 1: A primitive for generating dummy data instances.

Input: D: Dataset, n: Dataset Size, d: No. Features
. **Result:** $\hat{D}$: Shuffled Data Instances
begin
  $\text{MAX}, \text{MIN} = \text{get\_range}(D, n, d)$
  $\hat{D} = D$
  while $|\hat{D}| < (n + L)$ **do**
    $v = \text{array}(d)$ // Initialization
    for $i \in \{0, d\}$ **do**
      $v[i] = \text{random}(\text{MAX}_i, \text{MIN}_i)$
    $\hat{D} \leftarrow v$
  return oblivious_shuffle($\hat{D}$)

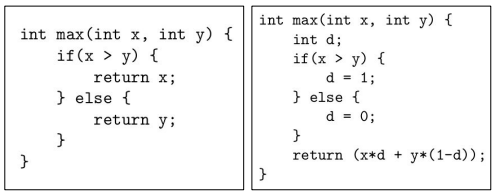


Fig. 2: Illustration of data-oblivious comparison.

ing our defense mechanism. We generate L dummy data instances and initially append them to the set of user-given data instances, forming $\hat{D}$. We then shuffle $\hat{D}$ in an oblivious manner, and sequentially process each data instance from the shuffled dataset during evaluation. One corner case is when `MIN` = `MAX`. With the goal of increasing variance of each feature in $\hat{D}$, we add an appropriate margin to `MAX` such that `MIN` < `MAX` is always true. In §4, we present the implementation details of oblivious data shuffling.

Data-Oblivious Comparison. We use a data-oblivious comparison primitive for checking whether a data instance is dummy or not. Typically, we first compute using a data instance, and then decide whether to ignore or retain the result of such computation depending on the type of data instance involved. We only desire to ignore results involving dummy data instances in a data-oblivious fashion. This ensures that the attacker observes resource access traces from both user-given and dummy data instances, which are indistinguishable.

Figure 2 illustrates the difference between non-oblivious and oblivious `max` function as an example of comparison primitive. Figure 2b is oblivious at the element-level since both conditional branch statements access the same set of variables. Whereas, Figure 2a is non-oblivious since either x or y is accessed when the `max` function returns depending on the conditional statement executed. In the case of an array, we access all elements in the array sequentially to remain data-oblivious. The mechanism proposed in [26] uses a more efficient compiler-based approach to perform oblivious comparison and array access at cache-level

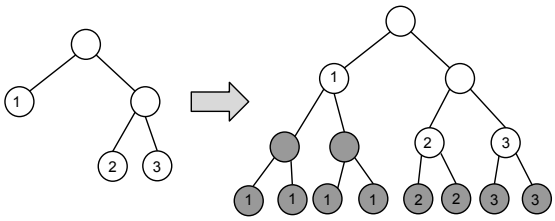


Fig. 3: Creating an obfuscated decision tree. Shaded nodes are formed using dummy data while others are formed using user-given data. Labels (denoted by $\{1, 2, 3\}$) of the original tree's leaf node is replicated in its descendant leaf nodes of the obfuscated tree.

granularity instead of element-level granularity. We leave its adaptation to our proposed approach for future work.

3.4 Learning Algorithms

Decision Tree Classifier. It is a tree-based model that uses a information-theoretic measures for data classification. In training a popular variant called ID3 [9], a feature with the largest information gain, with respect to the class label, is selected for partitioning the dataset into disjoint subsets. By iteratively performing this data partitioning on each residual data subset, a tree structure is created. Each feature value used for partitioning (or rule) then becomes either the root or an internal node of this tree. A leaf is formed when further partitioning is discontinued or unnecessary, i.e., when either all features are used along a path from the root, all data instances within the residual data subset has the same class label, or a user-defined maximum tree height is achieved. The last stopping condition is typically used to reduce overfitting [9]. During evaluation, class label of a test data instance is predicted as the majority label at a leaf that is encountered by following tree branches, starting from the root, according to its feature value consistent with the associated rule of intermediate tree nodes.

When a naive implementation of the above algorithm is employed within an SGX enclave, the attacker may track data-dependent tree node accesses during evaluation. This reveals the tree structure as well as the path of each test data instance. A typical strategy to defend against this side-channel inference-based attack is to balance the tree by adding dummy nodes, and access all nodes during evaluation of each test instance. As mentioned in [26], such a strategy has a runtime complexity of $\mathcal{O}(n\alpha)$ during evaluation, where α is the number of tree nodes. However, the complexity in a naive implementation is $\mathcal{O}(n \log \alpha)$. Clearly, data-obliviousness is achieved at the cost of computational efficiency, especially when α is large.

Instead, we utilize the dummy data generation primitive to obtain a contaminated dataset, and use the naive evaluation algorithm for class label prediction. During training, we learn a decision tree using user-given training data

instances (with known class labels), and create a balanced tree using dummy data instances, offline. Figure 3 illustrates an example of a balanced decision tree. Here, a tree (we term as *original*) resulting from user-given training data instances is obfuscated with nodes created from dummy data instances to obtain a balanced tree. Leaf nodes in the obfuscated tree reflect the class label of its ancestor node that form a leaf in the original tree. Clearly, the predicted class label of a test data instance on the obfuscated tree is the same as the original decision tree. Since dummy data instances are obliviously shuffled with user-given test data instances, access traces obtained by the attacker for dummy data instances are indistinguishable from that of user-given test instances. Therefore, the true data access path is hidden in the overall noisy access path obtained by the attacker. With L dummy data instances in the contaminated dataset, the time complexity of evaluating n user-given test data instances is $\mathcal{O}((n+L)\log\alpha)$.

Naive Bayes Classifier. It is a Bayesian model trained with an assumption of feature independence, given class labels [9]. Similar to the decision tree model, we train a Naive Bayes classifier offline with a user-given training dataset and evaluate test data instances online, i.e., within an SGX enclave. During evaluation, the predicted label of a test data instance is a class with the largest conditional probability, given its feature values. Such a classifier is typically used in the field of text classification that has large number of discrete valued features. The product of class conditional probability is computed for each feature value of user-given test data instance. Naively, one can pre-compute conditional probability for each feature value during training and access appropriate values during evaluation. In this case, an attacker may infer class and feature proportions of a given test dataset by tracking access sequence of pre-computed values. In a purely data-oblivious defense strategy, every element in the pre-computed array is accessed for evaluating each test data instance. If each of the d features have a discrete range of size f, computational time overhead for evaluation is $n \times d \times f$, whereas that of the original naive evaluation is $n \times d$. Clearly, this is a bottleneck in execution time when the range f is large. Instead, we utilize our dummy data generation primitive during evaluation by employing the naive method for accessing pre-computed array elements, inducing access patterns that are alike for both user-given and dummy data instances. The overhead in computational time for our modified version of Naive Bayes is $(n+L) \times d$. If $L \ll f$, our proposed defense is more efficient than the pure data-oblivious solution.

K-Means Clustering. The goal of k-means clustering is to group data instances into k disjoint clusters, where each cluster has a d-dimensional centroid whose value is the mean of all data instances associated with that cluster. Clusters are built in an iterative fashion. We follow a streaming version of Lloyd's method [9] for constructing clusters and evaluating user-given test data instances, since they are suitable for handling large datasets. During training, k cluster centroids are created by iteratively evaluating its value with least mean squared Euclidean distance, and re-evaluating cluster association of user-given

data instances using the computed centroid. Evaluation is performed online, i.e., within an SGX enclave. The user provides learned centroid and a set of test data instances. While cluster association of each data instance is evaluated by computing the minimum Euclidean distance to centroids, we re-compute the centroid of its associated cluster using the test data instances.

In a naive implementation of k-means clustering, the attacker can infer sensitive information, such as cluster associated of each data instance by tracking the centroid being accessed during assignment, and cluster proportions during centroid re-computation. The pure data-oblivious solution addresses this problem by performing dummy access to each centroid. On the contrary, we utilize the dummy data generation primitive to perform cluster assignment of both dummy and user-given data instances in an oblivious manner, and use the unmodified naive cluster re-computation method. This adds noise to cluster proportions inferred by the attacker. Since the number of clusters is fixed and is typically small, the time complexity remains the same as the original algorithm [26].

4 Implementation

One possible attack on the proposed defense strategy is to collect access traces of identical test data instances during evaluation, and use a statistical method to identify execution pattern of user-given test data instances in them. The main idea is that though these traces will be poisoned with execution involving random dummy data instances, execution of identical test data instances remain same. An attacker may produce such identical test instances by capturing an encrypted user-given instance at the application side, and providing identical copies of this data as input to the enclave application. We use a simple technique for discouraging this replay-based statistical attack by associating each data instance with a unique ID (called *nonce*), whose value is generated from a sequential counter. When data instances are passed to the enclave in response to an `ocall`, we check for data freshness within the enclave by comparing the internal nonce state to the nonce of each input. We proceed with evaluation if each new nonce value is greater than the previous one, else we halt execution. Since an attacker cannot change the nonce value of an encrypted data instance, this can detect stale instances used for a replay attack. We are aware that there exists superior methods for generating dummy data instances to thwart replay-based attacks in related domains [20], and leave its exploration for future work.

An important technique for reducing the effectiveness of inferring sensitive information from side-channels is the random shuffling of dummy data with user-given data instances in a data-oblivious manner. For simplicity, we assume that domain of each feature in the dataset is either discrete or continuous real-valued numbers. Nominal features are converted into binary vector using one-hot encoding [24]. Data shuffling is performed as follows. For brevity, we call the array containing data instances within the enclave as *data-array*. We associate a random number to each element of the data-array. Initially, dummy data instances are appended to the data-array as soon as they are created. We utilize

`sgx_read_rand` for random number generation. We then shuffle this array using an oblivious sorting mechanism over these random numbers. Similar to [26], we implement the Batcher's odd-even sorting network [5] for data-oblivious sorting, utilizing data-oblivious comparison during data swap when necessary. The runtime of this sorting method is $\mathcal{O}((n+L)(\log(n+L))^2)$. There are other shuffling algorithms with more efficient runtime complexity. We leave its applicability for future work. Meanwhile, we use a Boolean array, of size equal to the data-array, where value of each element indicates whether the corresponding instance in data-array is dummy or otherwise. Using oblivious comparison primitive, we identify and ignore computational results involving dummy data instances while sequentially evaluating the shuffled dataset.

5 Evaluation

Next, we analyze privacy guarantee of our proposed method and empirically evaluate computational overhead on various datasets.

5.1 Quantification of Privacy Guarantee

In our attack model, the attacker obtains execution traces in terms of sequential resource access while performing data analytics with user-given data instances. An attack on data privacy is successful when the attacker infers sensitive information from these traces by identifying distinguishing characteristics. However, the attack is unsuccessful if such distinguishing characteristics are either eliminated or significantly reduced via a defense mechanism. Such defenses are effective when they can provide quantifiable guarantees on data privacy. The primary question is how to measure privacy? Authors in [26] measure data privacy in terms of indistinguishability of a trace against a randomly simulated one. Since our defense mechanism primarily consists of performing non-essential or fake resource accesses, we define this indistinguishability in terms of *trace-variants* that is possible in a data analytics model. A trace-variant can be viewed as a sequence of page (or cache line) access when evaluating a test data instance. If N is the total number of trace-variants observed by an attacker from the model, we compute *Privacy-Guarantee* (denoted by γ) as the ratio of fake trace-variants to the total number of observed trace-variants. The value of N may depend on the variance in data and model. From a defense strategy perspective, every new data instance can provide a different access sequence at best. In this case, $N = n$ where n is the user-given dataset size. The following analysis assumes this case for simplicity, including the defense against replay attack mentioned in §4.

In a purely data-oblivious solution [26], there are $N-1$ fake trace-variants during evaluation since all possible cache-lines are accessed so that access pattern is the same for all data instances. For example, all nodes in a decision tree is accessed for evaluating the class label of each data instance. Here, each node may reside on a different cache-line or page. Therefore, $\gamma = \frac{N-1}{N}$. Note that $\gamma \simeq 1$ with large N; privacy is guaranteed on large N when this defense mechanism

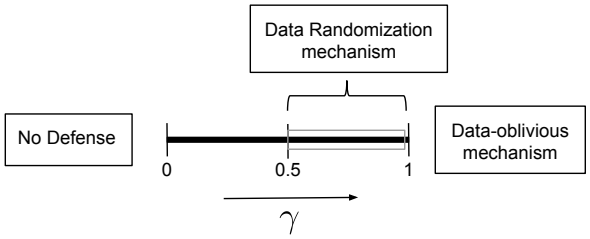


Fig. 4: Measuring privacy guarantee of SGX defense mechanisms.

is applied. On the other end of the privacy-guarantee spectrum, $\gamma = 0$ when no defense is applied, i.e., no fake trace-variants are possible. At this extreme, no privacy is guaranteed to the user's data. Figure 4 illustrates this privacy-guarantee spectrum.

Our proposed solution provides asymptotic privacy guarantee in terms of number of dummy data instances used. Since L dummy data instances are generated, there are at most L fake trace-variants with $N+L$ observed trace-variants. Therefore, the associated privacy-guarantee is $\gamma = \frac{L}{N+L}$. Clearly, a larger value of L provides greater privacy guarantee; it tends towards the γ value of purely data-oblivious solution (i.e., $\gamma \simeq 1$) for large L. If $L < N$, then an attacker can simply guess each trace to be true and infer sensitive information with a higher probability than random. Therefore, we choose $L \geq N$ to limit probability of a correct guess by the adversary to $\frac{1}{2}$ at best (as shown in Figure 4), similar to [26]. We now empirically demonstrate our proposed technique, and showcase the trade-off between privacy guarantee and computational efficiency with different choices of L.

5.2 Datasets

We measure execution time overhead of the proposed defense strategy using 3 publicly available real-world datasets [28] and a synthetic dataset. Table 3 lists these popular datasets with corresponding data statistics. The *Arrhythmia* dataset consists of medical patient records with confidential attributes and ECG measures. The problem is to predict the ECG class of a given patient record. The *Defaulter* dataset consists of financial records containing sensitive information regarding clients of a risk management company. The problem is to predict whether a client (i.e., a data instance) will default or not. Next, we use a benchmark dataset called *ForestCover*. Here, multiple cartographic attributes of a remotely sensed forest data are given. The problem is to predict forest type of a given data instance. Finally, we create the *Synthetic* dataset from a popular software for data stream mining called MOA [8].

These datasets may contain continuous and discrete valued features. For simplicity of implementation, we evaluate the decision tree and Naive Bayes classifiers using a quantized version of each dataset. We divide each feature range

Dataset	Statistics			Time Overhead					
	Size (n)	Features (d)	Classes (C)	Decision Tree		Naive Bayes		K-Means	
				SGX +Obliv	SGX +Rand	SGX +Obliv	SGX +Rand	SGX +Obliv	SGX +Rand
Arrhythmia (A)	452	280	13	52.49	**9.37**	319.15	**6.11**	**4.16**	6.36
Defaulter (D)	30,000	24	2	4.13	**1.11**	1.56	**1.10**	**1.07**	1.17
ForestCover (F)	50,000	55	7	2.72	**1.09**	3.13	**1.08**	**1.05**	1.07
Synthetic (S)	50,000	71	7	2.53	**1.09**	3.47	**1.07**	1.22	**1.09**

Table 3: Dataset statistics and empirical time overhead with $L = n$.

into discrete bins of equal width. For decision tree, we use $f = 10$ bins. However, for Naive Bayes, we use $f = 1000$ bins to reflect the dimensionality mentioned in §3.4. Nevertheless, we use the original form of each dataset to evaluate the k-means clustering algorithm.

5.3 Results and Discussion

The goal of empirical evaluation is to study and demonstrate applicability of our defense strategy in various settings. We implement a pure data-oblivious strategy, similar to [26], using data-oblivious comparison and array access over naive implementation of each data analytics algorithm. This baseline defense strategy is denoted by *Obliv*, whereas our proposed implementation is denoted by *Rand*. For each modified data analytics algorithm (i.e., Obliv and Rand), the computational *time overhead* is measured as the ratio of time taken by the modified algorithm executed within an SGX enclave to that of a naive implementation executed without SGX support. We perform all experiments on an SGX-enabled 8-core i7-6700 (Skylake) processor operating at 3.4GHz, running Ubuntu 14.04 system with a 64GB RAM.

Table 3 lists the time overhead measured on each dataset for decision tree and Naive Bayes classifiers, as well as k-means clustering, averaged over 5 independent runs. Note that we denote the defense strategies with $SGX{+}x$, where $x = \{\text{Obliv}, \text{Rand}\}$, to emphasize that they are executed within an SGX enclave. Since SGX currently supports limited enclave memory, we evaluate in a streaming fashion by dividing the dataset into small disjoint sets or chunks. Evaluation is performed over each chunk of size 64, over the given pre-trained model.

From the table, Rand clearly performs significantly better than Obliv in the case of decision tree and Naive Bayes classifiers. For example, Rand has only 11% overhead when class labels are evaluated using a decision tree in $16.76s$, compared to Obliv that takes $62.02s$, on the Defaulter dataset. When executing without any defense within the SGX enclave, it took $16.13s$. This shows that overhead due to enclave operations is small, as expected [17]. A higher overhead is observed in the Arrhythmia dataset due to smaller dataset size. For example, the naive implementation of decision tree on this dataset takes $0.01s$, compared to $0.79s$ in Obliv, and $0.14s$ in Rand. Also, it took $0.08s$ on the implementation within SGX enclave, but without employing any defense strategy. Clearly, the

cost of dummy data operations in Rand can be observed in the larger execution time compared to the naive implementation, yet it is much lower than Obliv.

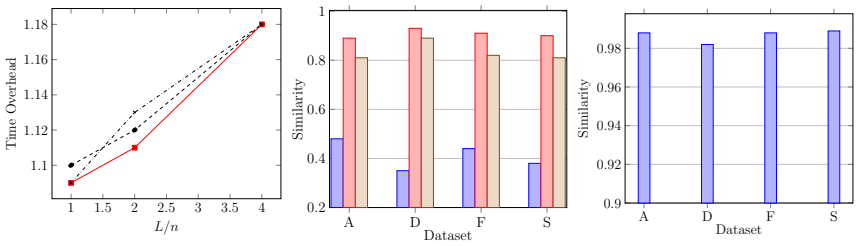
Limitations. For both decision tree and Naive Bayes classifiers, the number of fake resource access in Obliv is greater than that of Rand. Evaluating every test data instances in Obliv accesses each branch in a decision tree, and each of the $d \times 1000$ elements in the pre-computed probability array of Naive Bayes. Meanwhile, corresponding resource access in Rand is significantly small. However, when resource access patterns in both Obliv and Rand is similar during evaluation, the compromise on privacy with little or no trade-off in computational time of Rand is not very enticing. Time overhead shown in Table 3 for k-means clustering algorithm indicates one such example. Here, every cluster has to be accessed when searching for the nearest centroid to a given test data instance. While in Obliv, centroid re-computation of cluster assignment may be performed for each cluster, the time taken for oblivious shuffling of $n + L$ elements in Rand seem to surpass this re-computation time overhead. Except for the Synthetic dataset, Obliv outperforms Rand in all other datasets. In this situation, it is better to use Obliv defense strategy that guarantee better data privacy than the Rand strategy which provides a sub-optimal privacy guarantee.

Cost of More Privacy. The above results for Rand uses equal number of dummy and user-given data instances, i.e. $L = n$. If L is increased to provide better privacy according to §5.1, the cost of oblivious data shuffling, in terms of execution time, increases since $n + L$ data instances are to be shuffled. Figure 5a illustrates this increase in time overhead when using a decision tree classifier with Rand defense on various datasets as an example. This indicates that the value of L can be chosen appropriately by a programmer with desirable trade-off between computational overhead and data privacy. For example, a larger value of L for higher γ may be appropriate when the model has larger search space, similar to the Naive Bayes classifier discussed in this paper. In such cases, higher value of γ reduces the likelihood of dummy data instances producing unique patterns, with respect to user-given data instances.

5.4 Security Evaluation

The goal of our security evaluation is to empirically address the two main questions regarding Rand's data privacy guarantee; 1) Are access traces observed by the attacker randomized?, and 2) Are traces obtained from evaluating user-given and dummy data instances indistinguishable? Using Pin Tool [22], we generate memory access traces (sequence of read and write) of each classifier implementation when executing it in the SGX simulation mode. Here, we create 5 disjoint sets of 16 randomly chosen data instances for each dataset.

To answer the first question, we obtain traces by independently evaluating the 5 sets of data instance on a classifier, for each dataset. We perform different experiments on classifier implemented with no defenses (naive), Obliv, and Rand,



(a) Time overhead. (b) Across similarity. (c) Within similarity.

Fig. 5: (a) shows time overhead with increasing L (in proportion of n) on decision tree classifier with Rand, on D, F and S datasets. (b) shows similarity scores between access traces across different sets of instances when evaluated on the same classifier. Here, comparison between different defenses are shown, i.e., Rand, Obliv, and no defense. Finally, (c) shows similarity between traces of user-given and dummy data instances within a set of instances evaluated on Rand.

for comparison. We then compute Levenshtein similarity [25], as a surrogate to measure noise addition, between traces from the 5 sets on each dataset. Here, more similarity implies less randomization (i.e., added noise). Figure 5b shows an example result on trace comparisons obtained by evaluating a decision tree with corresponding defenses. In the figure, we can observe that traces from Obliv are more similar to each other (across the 5 sets) than those from the naive implementation, as mentioned in [26]. For example, in the Arrhythmia dataset, we obtain a similarity measure of 0.89 for Obliv compared to 0.81 for naive. However, traces from Rand are more dissimilar to each other compared to Obliv and naive approaches, indicating more data variance and randomization. On the contrary, we address the second question by comparing traces within a single set of 16 data instances. Concretely, we compute Levenshtein similarity between traces obtained by evaluating user-given data instances only, and those of dummy data instances only, in each set. Figure 5c illustrates an example on decision tree classifier with Rand. The high similarity scores between traces corresponding to the two types of data instances indicate indistinguishability.

6 Related Works

Studies on applications using Intel SGX have focused on an untrusted cloud computing environment. The first study in this direction [7] executed a complete application binary within an enclave. However, using this method on applications requiring large memory caused excessive page-faults that revealed critical information [32], thereby violating data privacy. To address this challenge, a recent study [29] used Hadoop as an application to split its interacting components between SGX trusted and untrusted regions. The main idea was to reduce TCB memory usage within the enclave for decreasing page faults. Challenges

in executing data analytics within an SGX enclave was first recently described by Ohrimenko et al. [26]. They propose a pure data-oblivious solution to guarantee privacy at cache-line granularity. We have compared our approach with a similar defense strategy. Alternative to algorithmic solutions, studies have proposed mechanisms to detect and prevent page faults attacks via malicious OS verification [12] and transactional synchronization [30].

A large group of studies in privacy preserving mechanisms deal with designing algorithms to preserve data privacy before data is shared with an untrusted environment [1]. Particularly, these studies focus on problems where identification of individual records are undesirable. Typically, the data is modified by addition of noise to features, regularization conditions, use of anonymization [10], and randomization [18] techniques. Instead, we focus on using a trusted hardware environment to protect privacy by using cryptographic methods to maintain confidentiality and trustworthiness [6]. We randomize side-channel information rather than user data for preserving privacy.

Use of dataset contamination to defend against adversaries is not new in machine learning settings. Studies on anomaly detection and intrusion detection [16] have discussed various types of attacks and defenses with regard to poisoning a user-given dataset with random data [4]. Particularly, a process called *Disinformation* is used to alter data seen by an adversary as a form of defense. This corrupts the parameters of a learner by altering decision boundaries in data classification. The process of *randomization* is used to change model parameters to prevent an adversary from inferring the real parameter values. These methodologies, however, limit the influence of user-given data in the learning process and may affect model performance on prediction with future unseen data instances. In all these cases, the adversary does not have control over the execution environment, and is weak. We instead leverage the effect of randomization to defend against side-channel attacks from a powerful adversary while performing data analytics on an Intel SGX enabled processor.

7 Conclusion

In this paper, we introduce a method to randomize side-channel information observed by a powerful adversary when performing data analytics over a SGX-enabled untrusted third-party server. With the help of dummy data instances and oblivious mechanisms, we study the trade-off between computational efficiency and data privacy guarantee in setting with large parameters. Our empirical evaluation demonstrates significant improvement in execution time compared to state-of-the-art defense strategy on data classification and clustering algorithms, with a small trade-off in privacy.

Acknowledgments. This research was supported in part by NSF awards CNS-1564112 and CNS-1629951, AFOSR award FA9550-14-1-0173, and NSA award H98230-15-1-0271. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily of the funding agencies.

References

1. Aggarwal, C. C., and Philip, S. Y. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-preserving data mining*. Springer, 2008, pp. 11–52.
2. Anati, I., Gueron, S., Johnson, S., and Scarlata, V. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy* (2013), vol. 13.
3. Antani, S., Kasturi, R., and Jain, R. A survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. *Pattern recognition 35*, 4 (2002), 945–965.
4. Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security* (2006), ACM, pp. 16–25.
5. Batcher, K. E. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference* (1968), ACM, pp. 307–314.
6. Bauman, E., and Lin, Z. A case for protecting computer games with sgx. In *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX'16)* (Trento, Italy, December 2016).
7. Baumann, A., Peinado, M., and Hunt, G. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS) 33*, 3 (2015), 8.
8. Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., and Seidl, T. Moa: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research* (2010), pp. 44–50.
9. Bishop, C. M. Pattern recognition. *Machine Learning 128* (2006), 1–58.
10. Brickell, J., and Shmatikov, V. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 70–78.
11. Costan, V., and Devadas, S. Intel sgx explained. *IACR Cryptology ePrint Archive 2016* (2016), 86.
12. Fu, Y., Bauman, E., Quinonez, R., and Lin, Z. Sgx-lapd: Thwarting controlled side channel attacks via enclave verifiable page faults. In *20th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)* (2017).
13. Gentry, C., et al. Fully homomorphic encryption using ideal lattices. In *STOC* (2009), vol. 9, pp. 169–178.
14. Götzfried, J., Eckert, M., Schinzel, S., and Müller, T. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security* (2017), ACM, p. 2.
15. Gupta, V., Lehal, G. S., et al. A survey of text mining techniques and applications. *Journal of emerging technologies in web intelligence 1*, 1 (2009), 60–76.
16. Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence* (2011), ACM, pp. 43–58.
17. Karande, V., Bauman, E., Lin, Z., and Khan, L. Sgx-log: Securing system logs with sgx. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (2017), ACM, pp. 19–30.

18. Kargupta, H., Datta, S., Wang, Q., and Sivakumar, K. On the privacy preserving properties of random data perturbation techniques. In *Data Mining, 2003. Third IEEE International Conference on* (2003), IEEE, pp. 99–106.
19. Lee, S., Shih, M.-W., Gera, P., Kim, T., Kim, H., and Peinado, M. Inferring fine-grained control flow inside sgx enclaves with branch shadowing. *arXiv preprint arXiv:1611.06952* (2016).
20. Li, F., Sun, J., Papadimitriou, S., Mihaila, G. A., and Stanoi, I. Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on* (2007), IEEE, pp. 686–695.
21. Liu, C., Wang, X. S., Nayak, K., Huang, Y., and Shi, E. Oblivm: A programming framework for secure computation. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 359–376.
22. Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. Pin: building customized program analysis tools with dynamic instrumentation. In *ACM Sigplan Notices* (2005), vol. 40, ACM, pp. 190–200.
23. Masud, M. M., Gao, J., Khan, L., Han, J., and Thuraisingham, B. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (2008), IEEE, pp. 929–934.
24. Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.
25. Navarro, G. A guided tour to approximate string matching. *ACM computing surveys (CSUR) 33*, 1 (2001), 31–88.
26. Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., and Costa, M. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium* (2016), pp. 619–636.
27. Rane, A., Lin, C., and Tiwari, M. Raccoon: closing digital side-channels through obfuscated execution. In *24th USENIX Security Symposium (USENIX Security 15)* (2015), pp. 431–446.
28. Repository, U. M. L. https://archive.ics.uci.edu/ml/datasets/, 1998.
29. Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. Vc3: trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 38–54.
30. Shih, M.-W., Lee, S., Kim, T., and Peinado, M. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA* (2017).
31. Shinde, S., Chua, Z. L., Narayanan, V., and Saxena, P. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (2016), ACM, pp. 317–328.
32. Sinha, R., Rajamani, S., Seshia, S., and Vaswani, K. Moat: Verifying confidentiality of enclave programs. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1169–1184.
33. Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., and Devadas, S. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 299–310.
34. Xu, Y., Cui, W., and Peinado, M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 640–656.