WASP-SC: Workload Agnostic Statistical Privacy against Side Channels

Abstract

In this paper, we present a novel hardware design for thwarting side channels on shared architectural resources which achieves a configurable level of privacy against an adversary observing hardware resource contention on a shared system. Our system creates a model of the program workload and resource utilization over time from an adversary's point of view by capturing the timing and frequencies of resource requests over a diverse set of inputs. We use this model and apply statistical noise to obfuscate the perceived contention for these hardware resources by probabilistically shaping the program behavior to be increasingly similar to other inputs. Doing so drastically reduces the probability that an adversary will be able to correctly infer information about the sensitive inputs to the program. Based on our simulation results, our methodology confers a tunable level of privacy, capable of reducing the accuracy of our simulated intelligent adversary to be no better than randomly guessing. Further, compared to a baseline IPC slowdown of 2.7-7x, our scheme incurs only 1.5-3x.

1 Introduction

In recent years, cloud platforms such as Amazon EC2 [1], Microsoft Azure [3], and Digital Ocean [4] have become increasingly popular for offloading intensive data and computational tasks. To increase performance and efficiency, providers may place multiple cloud tenants on the same physical machine within separate virtual machines (VMs). With this rise of third-party-operated cloud platforms, cloud security has become a pressing issue. However, concerns arise as the end-users do not know how or where their private data is being stored.

In scenarios such as this, it has been shown that sensitive information can be leaked through architectural side channels [9]. By measuring usage of resources in the architecture (e.g. caches, predictors, queues, etc.) during program execution, an adversary is able to glean this sensitive information. While the prospect of a malicious, co-resident adversary contending for resources is a possibility, a more compromising scenario would be that of a malicious hypervisor with the ability to measure fine-grained resource usage through hardware performance counters.

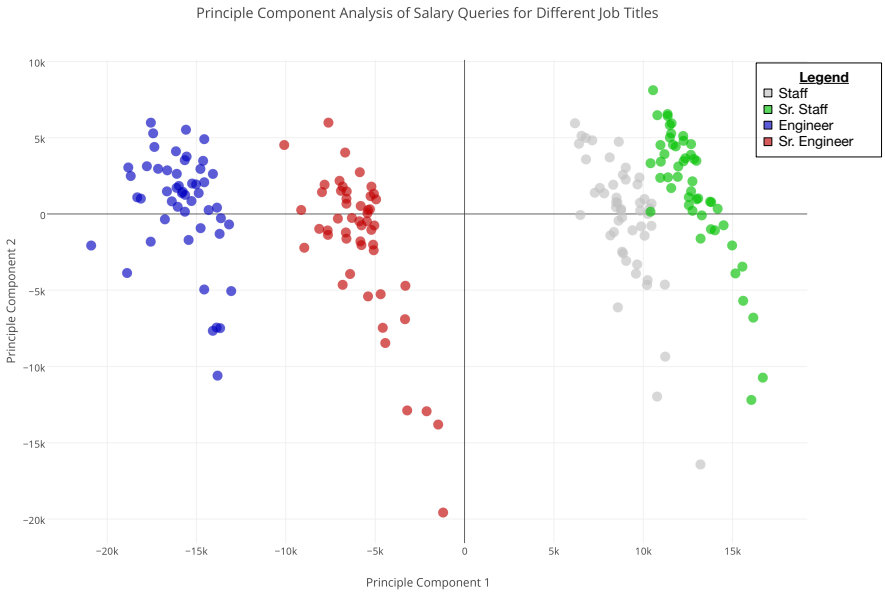

Figure 1: Principal Component Analysis of time-series of off-chip DRAM requests for various salary queries in MySQL based on employee title, recorded using Perf [14] on Linux.

Many architectural channels which involve information leaks across cores can be avoided by partitioning the underlying hardware resource. Lower level caches, for example, can be partitioned between cores to mitigate L1 cache timing attacks. However, some architectural resources may not be feasible to partition. Hardware queues, for example, are common shared resources that may be impractical to partition, but information may be leaked by the level of contention between two competing hardware threads which both access this queue. This scenario motivates a strong defense for *queue-based contention channels*.

A possible approach to thwarting contention channel attacks for shared resources is to time division multiplex (TDM) the channel. This provides the perception of having a physically partitioned resource without investing the space and power for the actual partitioning. However, this approach scales extremely poorly when dealing with a large number of hardware threads. Applying an approach of this nature on many different hardware queues simultaneously would simply be infeasible. In addition, queues that reside outside an individual core are shared by many hardware threads in large multi-core systems. Another approach would be to multiplex the queue based on space by setting a maximum number of elements a thread could occupy in the queue.

Partitioning channels in this manner can only protect from a malicious co-

located attacker, however. Ristenpart et. al demonstrated that it is possible for a malicious attacker to force co-location on these cloud platforms [21], but this is still only a small part of the attack scenario. A more feasible and stronger attack scenario is that of a malicious hypervisor [27] attempting to gather information about program workloads. A malicious hypervisor has easier access to more channels through which information may leak, such as hardware performance counters that directly measure events such as cache misses and off-chip DRAM requests.

Since we cannot effectively apply a partition-based defense, we must therefore alter the perceived contention for the underlying resource. Solely modifying the hardware counters would not protect against a malicious co-resident utilizing timing attacks to measure resource utilization. In addition, accurate counters are vital to enabling users to optimize their workloads for the specific machine in the cloud. The remaining option is to change the behavior of the program to modify how often it accesses the resource over time.

In this paper, we propose a novel method to alter a program's usage of a shared resource to achieve a bounded measure of privacy against an intelligent attacker. By creating a statistical model of how frequently individual workloads access the resource over time, we can set an upper bound on how much information we leak through the rate of access at each time-step. This statistical model additionally simulates the information a well-equipped attacker may glean through the contention channel. A simple attack can be demonstrated by examining one such statistical model using performance counter values on an example employee database [18]. Figure 1 depicts the Principal Component Analysis (PCA) on the generated model for salary queries of employees with various job titles. From the PCA, we can see that simply graphing the two axes with the highest statistical significance (principal components) demonstrates the ease of determining the job title used in each individual query.

We apply our noise addition scheme to alter resource utilization of workloads in a provably private manner. With this scheme, we incur an IPC slowdown of between 0.4 and 0.5 for different privacy budgets. We additionally model an intelligent adversary capable of perfectly determining private workloads before the introduction of our noise scheme. Testing our trained attacker on the noisy traces for our different privacy budgets, we see its success drop to 11% and 55%. We additionally introduce a perfectly secure variant of our scheme which reduces IPC by a factor of roughly 0.25 but prevents any machine- learning based classification attacks through side channels of the protected resource.

Our major contributions in this paper are:

- A general hardware queue able to alter its perceived resource usage according to a user provided model.
- A scheme for *replaying* resource utilization when executing a private workload. This scheme allows us to perturb the exact amount of utilization to statistically bound any information leaks from accessing the resource.
- An *optimal* noise-addition scheme that provides *provable* privacy guaran-

tees for variable privacy budgets.

- A *median replay scheme* that enables *perfect anonymity* among a given set of inputs.

2 Background and Related Work

Side channel attacks can refer to any form of adversarial attacks which seek to leak information from a running program through the physical side-effects of its execution. Side channel attacks originally focused primarily on the theft of cryptographic keys to break encryption schemes, but the definition and scope of side channel attacks has greatly broadened with the increasing concern of data privacy. A subset of side channels, *contention channels* [7], involve information leakage between two parties through shared microarchitectural resources, such as caches and branch predictors. Contention channels have been studied in great depth, and many previous attempts have been made to thwart them.

A common defense for side channels is to partition resources so that there is not any perceived contention between competing virtual machines. Cache partitioning schemes for side channel attacks such as RPcache and PLcache have been proposed to mitigate cache-interference-based side channel attacks. Two classes of security-aware cache designs were proposed by Wang et al. [28] to defend against many variations of cache side-channel attacks. Partitioned-locked caches (PLcache) create dynamic partitions by locking individual cache lines that cannot be evicted by other cache acceses originating from other threads, preventing cache interference. The second approach, random permutation caches (RPcache) allow for sharing between separate threads but randomizes and limits the interactions between threads.

Düppel [32], presented by Zhang et al., provides a mechanism for periodically cleansing the L1 cache to mitigate side channels on time-shared caches. Using hrtimers and software timers generated by inter-process interrupts (IPIs), Düppel flushes the contents of the L1 cache periodically.

However, partitioning is not always possible and there may exist additional problems which remain in some of the architectures which employ partitioning schemes. For example, the partitioning and randomization schemes do not scale well with the number of executing hardware threads. This scheme additionally does not protect against contention channels that do not rely on cache line addresses. Additionally, Düppel does not effectively protect in cloud settings where SMT is enabled. Additionally, these schemes do not affect shared caches beyond L1.

Recent work in mitigating LLC side channels leverages Intel Cache Monitoring Technology (CAT) to form partitions. Intel CAT is a feature introduced with the Haswell microarchitecture that allows access to different portions of the cache based on a "class of service" (COS). As COS natively only supports four classes, and complete partitions may be inefficient, CATalyst [15] creates both secure and non-secure partitions and manages the secure portion of the

cache in software. The non-secure portion of the cache remains managed by hardware. The two main security claims of this scheme are that: malicious code cannot evict secure pages and secure pages from different security domains cannot overlap. However, this proposed solution still does not address other side channels.

Privatizing program execution via anonymization of virtualized hardware resources has been previous explored. Xiao et al. [30] thwart storage side channels (specifically procfs) by applying differential privacy to the file system itself.

Differential privacy is a statistical concept that has been extensively applied and adapted since it was first formulated by Dwork [5]. Differential privacy was originally intended to quantify the probability of information leakage as a byproduct of database queries. Speficially, a function which provides ϵ-differential privacy has the informal guarantee that it will produce provably "similar" output from two databases which differ on at most 1 element. The metric under which output similiary is determined may vary depending on the setting.

Prior studies have attempted to extend the definition and applications of differential privacy to scenarios beyond simple databases. One notable application is that of geo-location indistinguishability [2]. Some work has attempted to transform time-series information into a database format to query and apply differential privacy directly. Others (PASTE) [20] have attempted to rework the definition of differential privacy to better suit time-series data. However, privatizing finite streams of data in real time is a problem that is fundamentally better suited for time-series analysis than database query analysis.

3 Threat Model and Attack Simulation

Our proposed system provides a methodology for obscuring utilization and contention for shared architectural resources. In our specific setting, we consider and evaluate the frequency of usage of the shared "uncore Global Queue" which tracks requests to DRAM and other cores of a third-party owned system. Such an environment may be found in cloud computing scenarios where users are given compute time on a remote machine, and their task is executed in a virtual environment. If the hypervisor in this setting is not trusted, private information about workload inputs may be leaked through hardware counters and contention for shared architectural resources.

We specifically address this queue within the memory controller as existing strategies such as partitioning are infeasible due to increasing hardware threads and relatively low queue size. Recent Intel Xeon server processors have as many as 36 hardware threads sharing the "uncore Global Queue" which has less than 64 entries for tracking requests [13].

In our proposed system, side channels within the core are assummed to be addressed by prior work such as resource partitioning and software-managed scratchpads. This is feasible as state-of-the-art server processors have few hardware threads within the core (e.g. two in Intel processors or eight in IBM

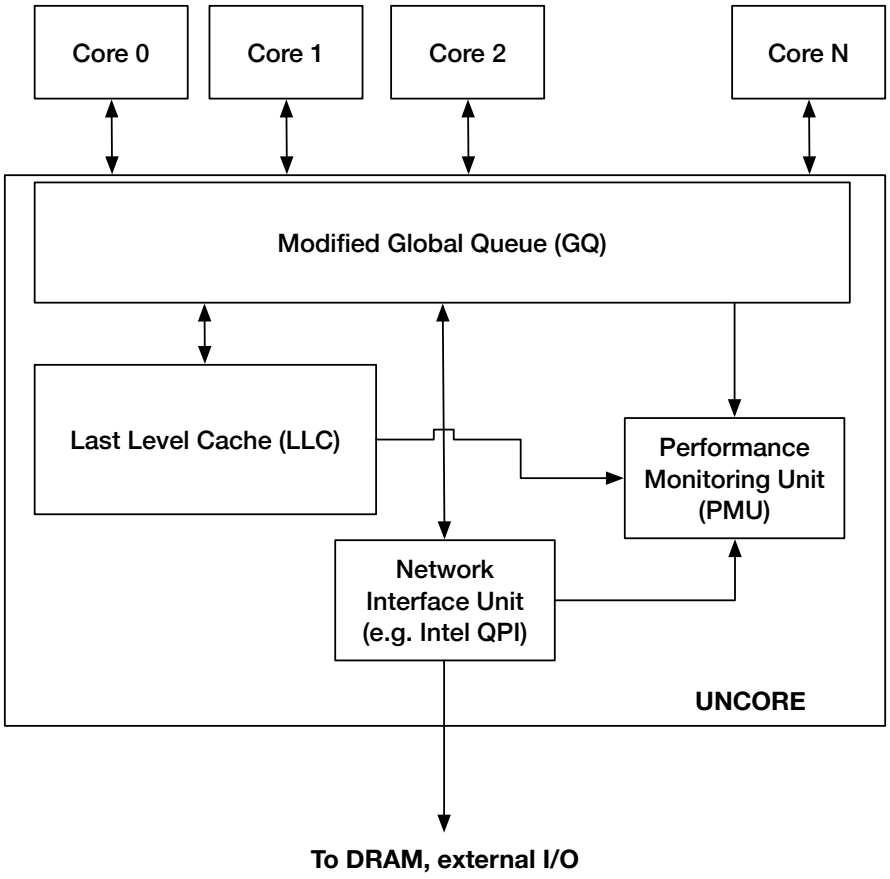

Figure 2: A victim

Power8). We also consider side channels after the memory requests are sent out from the "uncore Global Queue" to be handled by fixed service policies [25].

In our methodology, we consider contention for an architectural resource as a time series of usage frequency over time called a *trace*. This allows us to apply non-traditional approaches to measure and thwart information leakage. More specifically, this enables us to apply differential privacy to the time series derived from the contention channel. We can then create a probabilistic bound on how much information can be learned through this channel from such a time series.

To model a malicious attacker, we collect potential side channel information by querying hardware counters using Perf and collecting memory request frequency traces from ZSim [24]. Since a malicious hypervisor on an infected system has the ability to both read hardware counters during workload exe-

cution as well as measure contention indirectly, our system must reduce any potential classification attacks from either attack approach.

3.1 Classification Attacks

When modeling an intelligent adversary, we assume that both the adversary and the victim know the expected unperturbed *trace* from executing an input workload. We refer to the set of input workload-trace pairs as the "model." The adversary may then derive features and build a classifier fit to the model traces [31]. They may then predict the input workload by examining resource contention and running the extracted trace through the trained classifier. To account for the worst-case scenario, we assume in our attack simulations that the adversary is able to extract the *exact* trace (i.e. the exact time- series of architectural resource usage).

To determine the efficacy of our modeled attacker, we consider an attacker which has been trained on the model against original traces, the median of a workload's input traces, and noisy traces replayed for different values of epsilons. This test demonstrates a baseline for how well a reasonably well-trained attacker with the ability to effectively measure contention would fare against our noise scheme. We note that this is not a definitive measure of how well an adversary will perform against the different replay and noise schemes.

4 Mathematical Framework

The mathematical framework of differential privacy has been recently developed (see e.g. [6]) to allow queries to be answered from a database while still protecting individual data entries. Differential privacy requires that two (so-called adjacent) databases which differ in only one entry always produce statistically indistinguishable outputs. This hides the presence (or not) of a single data row in the database.

Inspired by this framework we enforce that our program leaks statistically indistinguishable traces for different inputs. This hides which input was used for this program from an adversary that observes a hardware channel leakage. Formally, consider a system S that takes an input x drawn *uniformly at random* from a set of possible inputs $\mathcal{X}$. The adversary observes $S(x)$, and tries to infer which input x was actually used.

Our goal is to design an output perturbation algorithm H, positioned between the adversary and $S(x)$, such that $H(x, S, \mathcal{X})$ and $H(x', S, \mathcal{X})$ are statistically indistinguishable for any two different inputs $x, x' \in \mathcal{X}$. We subsequently drop $S, \mathcal{X}$ from our notation and use $H(x)$ as the leaked information for input x.

Since the adversary can only observe $H(x)$ and not $S(x)$, if we enforce that $H(x)$ and $H(x')$ are close (for some notion of statistical distance), then it will be hard for the adversary to learn if the input x or x' was used through the hardware channel leakage. On the other hand, changing the output from $S(x)$

to $H(x)$ has a performance cost, since it requires artificial hardware activity. Our problem consists of minimizing the performance overhead while remaining within prescribed privacy guarantees.

We require the following for the leaked outputs:

Definition 1. *A system offers an ϵ level of privacy if for any two inputs $x, x' \in \mathcal{X}$ and any output $y \in \mathcal{Y}$ the probability distribution of the leaked output is within a factor of e^{ϵ}:*

$$Pr[H(x) = y] \leq e^{\epsilon} Pr[H(x') = y],$$
$$\forall x, x' \in X \quad , \forall y \in \mathcal{Y}.$$

The privacy parameter ϵ restricts how much we can perturb the outputs. Smaller values for ϵ give a higher privacy guarantee (output distributions have to be closer, for smaller ϵ) at the cost of added performance overhead. For example, when $\epsilon = 0$, Inequality 1 is satisfied only when $Pr[H(x) = y] = Pr[H(x') = y]$ for all x, x', y, meaning the output distributions of $H(x)$ and $H(x')$ have to be exactly the same. When ϵ is large a wide range of distributions are allowed and hence we have more freedom in how we perturb the outputs, hopefully incurring lower costs in the process. The privacy parameter ϵ also has a statistical interpretation: it can be used to guarantee a lower bound on the probability of error for any adversary trying to distinguish between different outputs [29, 17].

4.1 Cost Model

For simplicity, we first consider the case where the output is a single numerical value drawn from a set $\mathcal{Y}$ and extend it subsequently. For each input $x \in X$, the probability distribution $Pr[H(x) = y]$ is a vector of length $|\mathcal{Y}|$ with nonnegative entries that sums to 1. If we concatenate these vectors into one long vector, the privacy constraints of Def. 1 define a polytope where this vector must lie in. We are therefore looking for the distributions in this polytope that incur the smallest performance overhead. Every perturbation of the original values requires artificial hardware activity and hence performance loss. Furthermore, since $H(x)$ can be a performance counter value or another variable measuring activity, larger perturbations require more artificial hardware activity and hence more overhead. We therefore define the cost to be the mean absolute difference between the original and perturbed output values.

Definition 2 (Absolute-difference cost model)**.** *The expected cost incurred by an output perturbation algorithm under the* absolute-difference cost model *is given by*

$$\mathbf{E}\left[|S(X) - H(X)|\right]$$
$$= \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \sum_{y' \in \mathcal{Y}} Pr[H(x) = y'] \sum_{y \in \mathcal{Y}} Pr[S(x) = y]|y - y'|,$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the sets of possible inputs and outputs, S is the original system and H is the modified system with our output perturbation algorithm.

The typical approach for privacy is to add Laplacian noise to ensure that the privacy constraints of Def. 1 are satisfied. While this approach provides privacy guarantees, this does not minimize the expected cost under our cost model given the desired level of privacy. In this paper we instead explicitly minimize the performance overhead subject to the privacy constraints. We show that our cost minimization problem becomes a linear program over this polytope of privacy-preserving distributions. This allows us to search over the space of all distributions instead of simply adding Laplacian noise and hence obtain much lower performance overhead.

5 Noise Addition

Recall that we consider that the input x is uniformly selected from a set of m possible inputs $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$. The input can be fed into S (the original system) or H (system with our output perturbation algorithm added). The adversary observes $H(X)$ and tries to infer which input was fed into the system.

Let $Pr[S(x_i) = j] = p_{ij}$ and $Pr[H(x_i) = j] = q_{ij}$. The distributions for the original system p_{ij} are measured and our optimization is over the vectors q_{ij}. The problem of minimizing the expected cost subject to the privacy constraints can be written as:

$$\text{minimize:} \qquad \sum_{i \in [m], j \in \mathcal{Y}} q_{ij} \left(\sum_{j' \in \mathcal{Y}} p_{ik} |j - j'| \right) \tag{1}$$

$$\text{subject to:} \qquad q_{ij} \leq \mathrm{e}^{\epsilon} q_{i'j}, \text{ for all } i \in [m], i' \in [m], j \in \mathcal{Y} \tag{2}$$

$$\sum_{j \in \mathcal{Y}} q_{ij} = 1, \text{ for all } i \in [m], \tag{3}$$

$$q_{ij} \geq 0, \text{ for all } i \in [m], j \in \mathcal{Y} \tag{4}$$

The objective function 1 is simply the expected cost from Definition 2 without the constant multiplier $\frac{1}{|\mathcal{X}|}$. The constraints 2 are that the distributions lie in the polytope defined by the privacy constraints for this given ϵ while constraints 3 and 4 simply restrict the solutions to be probability mass functions. This optimization problem is a linear program (LP) in the q_{ij}'s. The output of our optimal perturbed system H is then simply a value sampled from the q_{ij}'s which can be obtained by solving this LP.

The (not necessarily unique) solutions of this LP have some interesting properties which we can exploit to optimize the computational efficiency and storage requirements when computing and storing the solutions. Denote the probability mass functions of $S(x_i)$ and $H(x_i)$ by P_i and Q_i respectively (so $P_i(j) = p_{ij}$ and $Q_i(j) = q_{ij}$). We show the following structural result about the solutions of this LP:

Proposition 3. *There is at least one optimal solution such that the union of the supports of $\{Q_i\}_{i\in[m]}$ is a subset of the union of the supports of $\{P_i\}_{i\in[m]}$.*

Before proving the proposition, we first prove three lemmas about an optimal solution that allow us to construct another optimal solution from it. Suppose there is an optimal solution $Q^* = \{Q_i^*\}$ in which $Q_i^*(j^*) > 0$ for some i, where $j^* \notin \bigcup_i supp(P_i)$. We construct another solution $\hat{Q} = \{\hat{Q}_i\}$ as follows.

1. Pick any $j^* \in \bigcup_i supp(Q_i^*) \setminus \bigcup_i supp(P_i)$.

2. Compute the value $\hat{j}$ as follows.

$$\hat{j} = \begin{cases} \min\{j : j \in \bigcup_i supp(P_i) \text{ and } j > j^*\} & \text{if } \sum_i q_{ij^*}^* \sum_{j>j^*} p_{ij} \geq \sum_i q_{ij^*}^* \sum_{j<j^*} p_{ij} \\ \max\{j : j \in \bigcup_i supp(P_i) \text{ and } j < j^*\} & \text{otherwise.} \end{cases} \tag{5}$$

   Intuitively, $\hat{j}$ is either the smallest support of $\{P_i\}$ that is larger than j^* or the largest support that is smaller than j^*. It is chosen in a way that minimizes the objective function when we move all the mass from $Q_i^*(j^*)$ to $Q_i^*(\hat{j})$.

3. Construct $\hat{Q}$ from Q^* as follows. For all i,

$$\hat{Q}_i(j) = \begin{cases} Q_i^*(\hat{j}) + Q_i^*(j^*) & \text{if } j = \hat{j} \\ 0 & \text{if } j = j^* \\ Q_i^*(j) & \text{otherwise.} \end{cases} \tag{6}$$

   Intuitively, we are moving all the mass from $Q_i^*(j^*)$ to $Q_i^*(\hat{j})$.

Lemma 4. *$\hat{Q}$ satisfies constraints 2-4 of the LP.*

Proof of Lemma 4. Satisfying constraints 3 and 4 is trivial, as we simply moved the mass from $Q_i^*(j^*)$ to $Q_i^*(j')$. To see why constraints 2 are still satisfied, consider the following cases.

Case 1. $[j = \hat{j}]$ For any $i \in [m]$ and $i' \in [m]$, we have

$$\begin{aligned} \hat{q}_{ij} &= q_{i\hat{j}}^* + q_{ij^*}^* \\ &\leq \mathrm{e}^{\epsilon} q_{i'\hat{j}}^* + \mathrm{e}^{\epsilon} q_{i'j^*}^* \\ &= \mathrm{e}^{\epsilon} \hat{q}_{i'j}. \end{aligned}$$

Case 2. $[j = j^*]$ Since $\hat{q}_{ij} = 0$, the constraints are trivially satisfied.

Case 3. $[j \notin \{\hat{j}, j^*\}]$ Since $\hat{q}_{ij} = q_{ij}^*$, the constraints are still satisfied.

□

Lemma 5. *Evaluating the objective function 1 on $\hat{Q}$ will not yield a higher value than evaluating it on Q^*. In other words,*

$$\sum_{i\in[m],j\in\mathcal{Y}} \hat{q}_{ij}\left(\sum_{j'\in\mathcal{Y}} p_{ij'}|j-j'|\right) - \sum_{i\in[m],j\in\mathcal{Y}} q^*_{ij}\left(\sum_{j'\in\mathcal{Y}} p_{ij'}|j-j'|\right) \leq 0.$$

Proof of Lemma 5.

$$\begin{aligned}
&\sum_{i\in[m],j\in\mathcal{Y}} \hat{q}_{ij}\left(\sum_{j'\in\mathcal{Y}} p_{ij'}|j-j'|\right) - \sum_{i\in[m],j\in\mathcal{Y}} q^*_{ij}\left(\sum_{j'\in\mathcal{Y}} p_{ij'}|j-j'|\right) \\
&= \sum_{i\in[m],j\in\mathcal{Y}} (\hat{q}_{ij}-q^*_{ij})\left(\sum_{j'\in\mathcal{Y}} p_{ij'}|j-j'|\right) \\
&= \sum_{i\in[m]} q^*_{ij^*}\left[\left(\sum_{j'\in\mathcal{Y}} p_{ij'}|\hat{j}-j'|\right) - \left(\sum_{j'\in\mathcal{Y}} p_{ij'}|j^*-j'|\right)\right] && \text{(Equation 6 in Step 3)} \\
&\leq 0. && \text{(Equation 5 in Step 2)}
\end{aligned}$$

□

The following lemma follows from above.

Lemma 6. *$\hat{Q}$ is an optimal solution to the LP.*

Proof of Lemma 6. By Lemma 4, $\hat{Q}$ will not violate constraints 2-4 of the LP. By Lemma 5, $\hat{Q}$ still minimizes the objective function. Therefore, $\hat{Q}$ is an optimal solution to the LP. □

We now make use of the above lemma to prove our proposition.

Proof of Proposition 3. Given an optimal solution $Q^* = \{Q^*_i\}$ in which $Q^*_i(j^*) > 0$ for some i, where $q^* \notin \bigcup_i supp(P_i)$, we can always construct another optimal solution $\hat{Q}$ that has the desired property, using the above algorithm with the following additional step.

4. Set $Q^* \leftarrow \hat{Q}$. If $\bigcup_i supp(Q^*_i) \setminus \bigcup_i supp(P_i)$ is the empty set, output Q^* as the solution. Otherwise, repeat from Step 1.

By Lemma 6, at the end of each iteration of the algorithm, Q^* is still an optimal solution to the LP. Note that j^* is removed from $supp(Q^*_i)$ and $|\bigcup_i supp(Q^*_i) \setminus \bigcup_i supp(P_i)|$ decreases by 1 in Step 3, so the algorithm will terminate. Finally, when the algorithm terminates, we have $\bigcup_i supp(Q^*_i) \setminus \bigcup_i supp(P_i) = \emptyset$, which means the union of the supports of $\{Q^*_i\}$ is a subset of the union of the supports of $\{P_i\}$, and thus Q^* becomes an optimal solution to the LP with the desired property. □

If the original output distributions corresponding to each input are deterministic (i.e., for any input x_i, $P_i(j) = 1$ if $j = \alpha_i$ for some value $\alpha_i \in \mathcal{Y}$, and $P_i(j) = 0$ otherwise), then the following corollary follows from Proposition 3:

Corollary 7. *Suppose $S(x_i) = \alpha_i$ where $\alpha_i \in \mathcal{Y}$. Let $\mathcal{A}$ be the set $\{\alpha_i\}_{i\in[m]}$. Then there is at least one optimal solution such that the union of the supports of $\{Q_i\}$ is a subset of $\mathcal{A} = \{\alpha_i\}$.*

For the case of two inputs, given α_1 and α_2, we can even solve for the q_{ij}'s directly.

Corollary 8. *When $m = 2$, given α_1 and α_2, we can compute an optimal solution $\hat{q}$ to the LP as follows.*

$$\hat{q}_{ij} = \begin{cases} \frac{e^\epsilon}{1+e^\epsilon} & \text{if } j = \alpha_i \\ \frac{1}{1+e^\epsilon} & \text{if } j = \alpha_{(i+1)\%2} \\ 0 & \text{otherwise.} \end{cases}$$

Proof. It is easy to verify that $\hat{q}$ satisfies constraints 2-4. We prove that $\hat{q}$ is indeed an optimal solution to the LP by first giving a lower bound for the objective function 1 evaluated at an optimal point q^*, and then showing that we meet this lower bound if we evaluate the objective function at $\hat{q}$.

Since $S(x_1) = \alpha_1$ and $S(x_2) = \alpha_2$, we have

$$p_{1,j} = \begin{cases} 1 & \text{if } j = \alpha_1 \\ 0 & \text{otherwise} \end{cases} \text{ and } p_{2,j} = \begin{cases} 1 & \text{if } j = \alpha_2 \\ 0 & \text{otherwise.} \end{cases}$$

From Corollary 7, we know that there is at least one optimal solution such that $q_{ij} > 0$ only if $j = \alpha_1$ or $j = \alpha_2$. Let q^* be one such solution. From the privacy constraints 2, we have

$$\begin{aligned} q^*_{1,\alpha_1} &\leq \mathrm{e}^\epsilon q^*_{2,\alpha_1}, \text{ and} \\ q^*_{2,\alpha_2} &\leq \mathrm{e}^\epsilon q^*_{1,\alpha_2}. \end{aligned}$$

The constraints 3 give $\sum_j q^*_{1,j} = q^*_{1,\alpha_1} + q^*_{1,\alpha_2} = 1$ and $\sum_j q^*_{2,j} = q^*_{2,\alpha_2} + q^*_{2,\alpha_1} = 1$, so we can rewrite these privacy constraints as

$$\begin{aligned} q^*_{1,\alpha_2} + \mathrm{e}^\epsilon q^*_{2,\alpha_1} &\geq 1, \text{ and} \\ \mathrm{e}^\epsilon q^*_{1,\alpha_2} + q^*_{2,\alpha_1} &\geq 1. \end{aligned}$$

Summing these constraints gives us

$$\begin{aligned} (1 + \mathrm{e}^\epsilon) q^*_{1,\alpha_2} + (1 + \mathrm{e}^\epsilon) q^*_{2,\alpha_1} &\geq 2 \\ q^*_{1,\alpha_2} + q^*_{2,\alpha_1} &\geq \frac{2}{1 + \mathrm{e}^\epsilon}. \end{aligned}$$

We can obtain a lower bound for the objective function 1 by substituting q^* into it, which gives us

$$\begin{aligned}
&q^*_{1,\alpha_2}|\alpha_1 - \alpha_2| + q^*_{2,\alpha_1}|\alpha_2 - \alpha_1| \\
&= (q^*_{1,\alpha_2} + q^*_{2,\alpha_1})|\alpha_1 - \alpha_2| \\
&\geq \frac{2}{1+\mathrm{e}^{\epsilon}}|\alpha_1 - \alpha_2|.
\end{aligned}$$

Substituting

$$\hat{q}_{ij} = \begin{cases} \frac{\mathrm{e}^{\epsilon}}{1+\mathrm{e}^{\epsilon}} & \text{if } j = \alpha_i \\ \frac{1}{1+\mathrm{e}^{\epsilon}} & \text{if } j = \alpha_{(i+1)\%2} \\ 0 & \text{otherwise} \end{cases}$$

into the objective function 1, we will meet this lower bound while satisfying constraints 2-4, showing that this is indeed a solution to the LP. □

For the case of $m > 2$ inputs, solving for the optimal q_{ij}'s is again a linear programming problem as above but with $q_{ij} = 0$ when $j \notin \mathcal{A}$ (by Corollary 7). Note that the number of constraints we have is usually much smaller since $m << |\mathcal{Y}|$. Moreover, if $\mathcal{Y}$ is not a finite set (e.g., $\mathcal{Y}$ is the set of all integers $\mathbb{Z}$), we cannot solve the original problem, but we can solve this new LP problem which considers outputs only from the set $\mathcal{A}$.

The above properties help in reducing the computational complexity by drastically reducing the number of constraints (or in the case of having only two inputs, we can easily compute the non-zero q_{ij}'s in closed form). The above properties also help reduce storage requirements when storing the solutions as follows. Instead of storing the entire output set $\mathcal{Y}$ and distributions $\{Q_i\}$ (requiring $(m+1)|\mathcal{Y}|$ numbers in the naïve way), we need to store only $\mathcal{A}$ and q_{ij} for $i \in [m]$ and $j \in \mathcal{A}$ (requiring less than $m(m+1)$ numbers), which can be a huge saving when $m << |\mathcal{Y}|$.

To leverage the above savings for outputs that are not deterministic, one heuristic that seems to work well is to first approximate each output distribution by its median. After that we can leverage the above savings and compute an *approximate* optimal solution. The rationale behind approximating with the median is that the median minimizes the expected absolute difference from samples drawn from a distribution.

5.1 Extending to time series

We previously formulated the optimal perturbation problem when the output is a single value. We can naturally extend this formulation when the output is a time series. The simplest and sub-optimal way to achieve this is to model the output of $H(x)$ as independent across time and apply the previous solution in each step independently. This nave approach would require computing and storing approximately $m(m+1)T$ parameters which would be prohibitively complex.

We instead use a formulation that offers better performance and models time-dependencies. Denote the median of $S(x_i)_t$ as α_{it} and let α_i be the time series $(\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,T})$. Instead of storing the entire time series α_i, if the outputs are correlated across time, we can heavily compress this time series by approximating α_i with an autoregressive model with a small order k. This way, we can compress the T points of α_i into $k+1$ coefficients and the k starting points $(\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,k})$, at the cost of additional computations needed to generate predictions for α_i at runtime.

For the case of two inputs, we know from Corollary 8 that at each time step t, the *locations* j for which q_{ijt} is non-zero depends on $\alpha_{1,t}$ and $\alpha_{2,t}$, but the *values* of the non-zero values depends only on ϵ (and not on α or t). Therefore, we need to store only α_1, α_2 and $\frac{1}{1+e^{\epsilon}}$.

Note that all the different ways of modeling and compressing the time-series will never violate the privacy constraints since we are always constraining $H(x)$ to lie within the privacy preserving polytope. A poor model will only limit the number of representable distributions to a subset of the privacy polytope that is easier to describe and optimize over. Hence simpler models lead to bad performance but always respect the ϵ privacy guarantees.

For the case of $m > 2$ inputs, the values of q_{ijt} depend on the values of the medians α_{it}'s and will change across time. We might be tempted to compress the q_{ijt}'s as we did with the α_{it}'s, but one caveat here is that while modifying the α_{it}'s will not change the privacy guarantee, modifying the q_{ijt}'s *will*, and the constraints of our LP might not be satisfied anymore.

6 System Design

Core	Westmere-like OOO, 3.2GHz
L1 I/D Cache	16KB, 2-way set-associative, 4-cycle latency D-cache, 3-cycle latency I-cache
L2 Cache	512KB, 4-way set-associative, 7-cycle latency, inclusive, 128B line-size
Memory	DRAMSim2 Micron 8MB, 8 banks, 16 width

Table 1: Configuration of simulated core for medical database

Core	Westmere-like OOO, 3.2GHz
L1 I/D Cache	32KB, 4-way set-associative, single-cycle latency
L2 Caches	256KB, 16-way set-associative, 7-cycle latency, inclusive
L3 Cache	2MB, 16-way set-associative, 20-cycle latency, inclusive
Memory	DRAMSim2 Micron 8MB, 8 banks, 16 width

Table 2: Configuration of simulated core for Ligra and Moses

Resource utilization side channels can be obfuscated in multiple manners. Our design is a hardware implementation which aims to artificially alter the perceived contention for architectural resources rather than report virtualized, noisy counter values. This has the benefit of scaling to various side channel

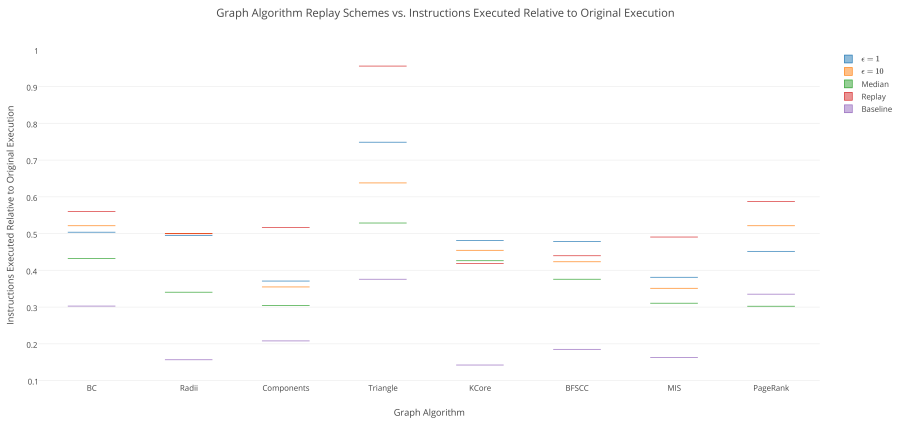

Figure 3: IPC slowdown for each of the 10 Ligra graph algorithms on 10 real-world graphs.

attacks such as timing, contention, and termination channels. Furthermore, simply reporting noisy hardware counter values does not prevent similar attacks being made by indirectly measuring usage of the same resource through other means, such as contention for the shared resource. For our experiments, we use the configurations shown in Table 1 and Table 2, which are our ZSim [24] configurations for the simulated processor.

As described in Figure 2, our experiments are concerned with protecting the side-channel resulting in the shared Global Queue of memory requests sent to the uncore by each hardware thread in the system. Rather than relying on strict partitioning or fixed scheduling of requests across all cores, we propose to replay a privatized model of the application's memory request behavior.

Our approach introduces a modified queue able to both throttle memory requests and introduce "dummy" requests based on a model of queue resource usage (the *trace* described in 3.) This trace is generated by the user to describe the frequency of memory requests sent to the queue at each timestep. Each value in the trace represents the number of requests sent to the queue every *interval* number of cycles. The number of cycles in an interval is determined by the specific system and workload the user is modeling (e.g. every 100,000 cycles.) Our modified queue ensures requests are sent *evenly spaced* within an interval, delaying real requests and adding "dummy" requests as appropriate. By evenly spacing requests, we eliminate the possibility of an unforseen information leak due to an attacker recording information at a finer granularity than our statistical model.

7 Experiments and Results

We evaluate a modified memory queue in ZSim [24] on three input sets. We simulate entire queries on a real-world medical database [16] and perform a classification-based attack as described in Section 7.1 to show how effectively our scheme obfuscates the attacker's observed memory request behavior. Further, we simulate a relevant portion of the execution of several large-scale benchmarks and report performance overheads as well as additional classification attack effectiveness. In both cases, we perform our evaluation on a set of representative program inputs (approximately 50) and begin simulation at the region-of-interest (ROI), such as the start of the PostgreSQL database query. We run our modeling experiments for each benchmark for 100 iterations and each noise addition scheme for 10 iterations.

7.1 Data Collection and Classification

We first modified the ZSim [24] memory controller that feeds requests to DRAMSim2 [22] to record the number of memory requests every *interval* cycles (200,000–500,000 in our experiments). This enables us to model an adversary measuring contention or utilization of the underlying shared queue. The memory request traces we derive from our ZSim simulations represent what a perfect adversary (such as a malicious hypervisor) would observe. The intelligent adversary that we model considers each indvidual input as a different *class* in the classification problem. As we test our adversary on multiple possible input workloads, the machine-learning problem is said to be a *multiclass classification* problem.

To perform the classification attack, we build a gradient boosting tree regressor using scikit-learn [19] and use it to predict workloads and inputs given memory trace time series. To build *feature vectors* for each input time series data, we first construct a set of reference median time series from each workload-input combination. We then populate the feature vector with Dynamic Time Warping (DTW) distance between each reference time series and each training time series [8]. DTW is used because it provides a more intuitive notion of "closeness" between two time series. Furthermore, we can account for the importance of the two time series being out of phase at different points. Additionally, our implementation of DTW allows us to compare time series which do not have the same number of data points, which is crucial for time series which describe the non-deterministic behavior of memory accesses. Lastly, we collect the first 20 wavelet coefficients for the discrete transform using the Daubechies 1 (db1) wavelet. The wavelet transform gives a representation of the time series that describes how and when the times series changes over time. This allows us to capture both frequency and timing information, something which is not possible from simply comparing time series with DTW, or by using other common transforms such as FFT or Laplace.

To train the classifier on our input matrix, we build the reference model described above using ZSim traces. The ZSim traces correspond to the precise resource utilization of the workloads over time and demonstrate the worst-case

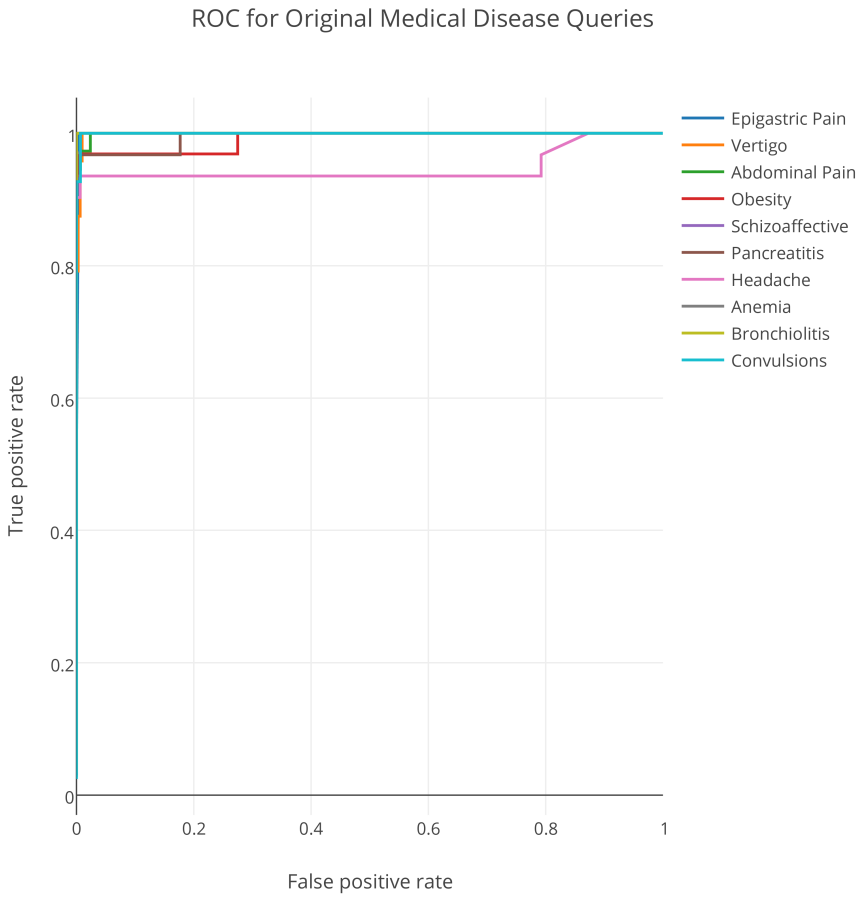

Figure 4: ROC curve for memory request classification on a subset of medical disease queries. Each query is for the average cost of different admitting patient diagnoses.

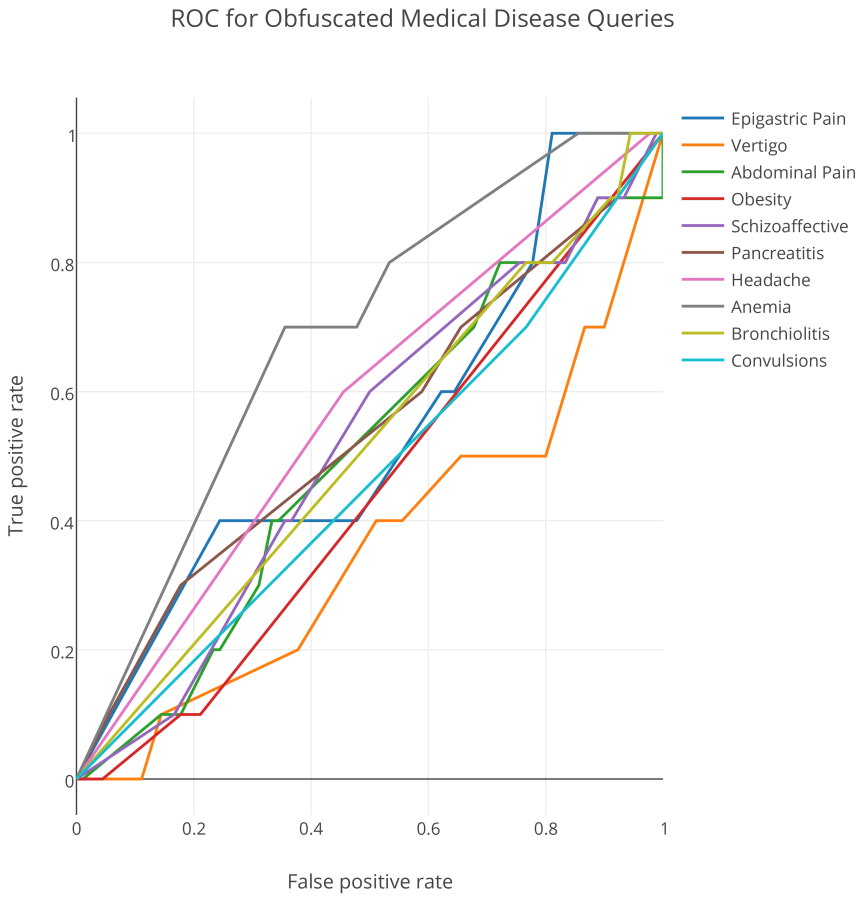

Figure 5: ROC curve for obfuscated memory request classification on a subset of medical disease queries using $\epsilon = 0.1$. Each query is for the average cost of different admitting patient diagnoses.

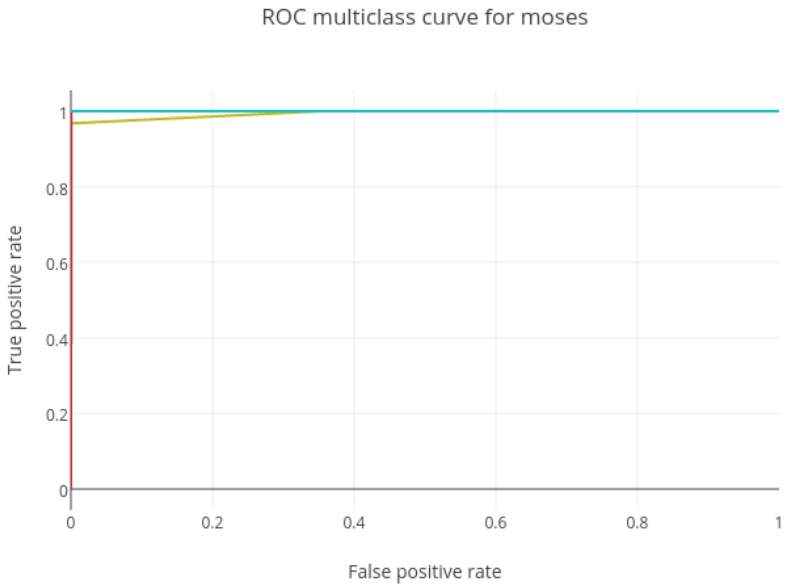

Figure 6: ROC Curve of memory request classification for Moses translation on 100 input sentences.

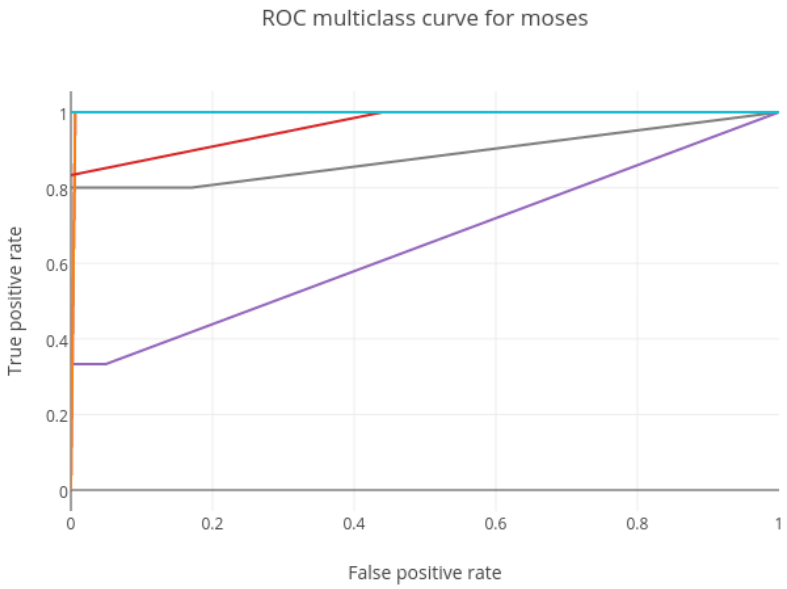

Figure 7: ROC Curve of memory request classification for Moses translation on 100 input sentences when replaying original trace. This demonstrates that while a significant overhead comes from replaying, there are no security benefits from doing so.

scenario of a perfect adversary with access to the exact resource utilization trace. This is the base offline knowledge we assume an attacker collects before performing the attack. We then use 5-fold cross validation to reduce the chance of overfitting error as well as the chance of fitting to any noise within the training data. When the model is built, the attacker may then apply the model to make predictions on the private workload (the *class*) when new runs are observed through the attack vector.

List 1: Ligra Graph Inputs

- Skitter Internet Topology
- Patent Citations
- Friendster Social Network
- Orkut Social Network
- Road Network California
- Road Network Pennsylvania
- Road Network Texas
- Live Journal Social Network
- Pokec Social Network
- Wikipedia Talk Network

For our main classification experiment, we simulate a PostgreSQL database containing one quarter of inpatient medical records in a Texas hospital from 2011. Our set of queries computes the average cost incurred by patients for their visit for 50 different International Statistical Classification of Diseases (ICD-9) codes. The ICD-9 codes represent the specific patient issue at the time of admittance (e.g. shortness of breath (786.09) is represented as 78609 in the database). Figures 4 and 5 show the ROC curve results of our classifier for a memory request trace with an interval size of 500,000 cycles before and after our noise addition scheme. The ROC curve is calculated for the multiclass case by graphing the true positive vs false positive rates for each individual class against the rest (*one vs all*) Our modeled attacker is able to classify the original trace with striking accuracy, while the probability of false alarms significantly increases when noise is added.

Next, we simulate 1 billion cycles of 10 graph algorithms on 10 real-world graphs from the SNAP datasets [12] shown in List 1. We utilize the Ligra graph processing framework [26] in single-threaded mode for our graph simulations. We also simulate a real-time translation system Moses [11] using a pre-built English-Spanish translation model based on the Europarl [10] language corpus

translating 50 sentences from Don Quixote.

Figure 3 shows the results of our ZSim runs of 48 Ligra algorithms and input combinations, grouped by algorithm. The overall trend is that of the replay trace having IPC closest to the original runs of the workload and median performing the worst. However, there are some exceptions due to different inputs and different algorithms not replaying well due to inherent randomness and nondeterminism.

7.2 Overhead Experiments

Next we show the IPC slowdown incurred for each of our three benchmarks: PostgreSQL medical db, Ligra (8 different graph algorithms), and the Moses real-time translation system. Since we simulate 1 billion cycles for each benchmark, we report the instructions executed relative to the original execution for each benchmark to show the IPC slowdown. Additionally, we compare our schemes to a *baseline* partitioning scheme which simulates the performance of partitioning the hardware resource using a time division multiplexing scheme.

We first examine the overhead associated with replaying the original trace, adding noise using our scheme, and replaying a secure version of the trace via the median. The results of this are shown in Figure 8. Replaying the original trace results in an IPC slowdown of roughly 0.5. Much of this can be attributed to evenly spacing out the memory requests within each interval to mitigate adversaries who may be out of phase or at a higher granularity than our model. As we perturb the trace more, the relative IPC further decreases, as is expected. However, we notice that the partitioning scheme we use as a baseline comparison has a significantly higher overhead than any of our schemes which intead add statistical noise.

Next, we determine the overhead incurred for privatizing inputs to Moses and graph the results in Figure 9. Similar results as what we see for the medical experiments. Lower amounts of perturbation to the trace correlate to a higher relative IPC. Additionally, the partitioning baseline has worse performance than the median trace, though the median trace is perfectly private.

8 Conclusion

Cloud privacy concerns are becoming an increasingly compelling issue with the rise of side channel attacks. Previous approaches have attempted to leverage a partitioning of hardware resources, but, as we have demonstrated, there are instances where partitioning becomes infeasible, namely shared hardware resources such as queues.

We present a method to obfuscate the effects of accessing a hardware queue by randomly adding artificial hardware activity. This results in leaked information that follows an artificially generated model of a program execution. We design this model to be both close to the real program (to minimize performance

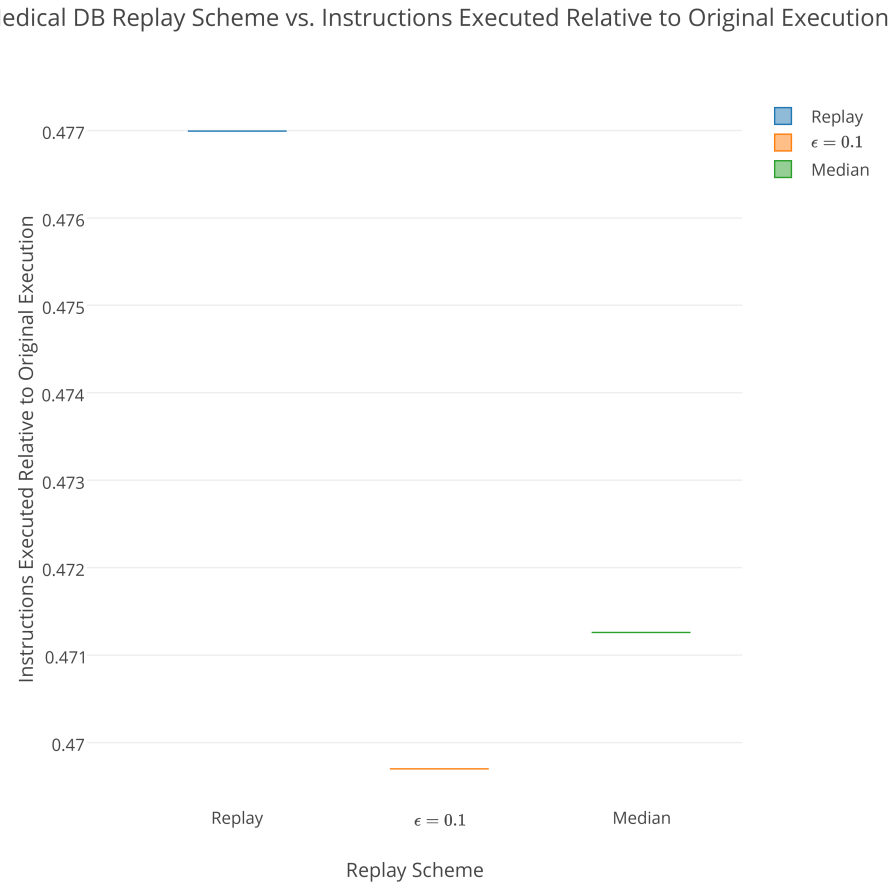

Figure 8: IPC slowdown across 50 different admitting patient diagnosis queries for 1 billion cycles for the four noise-addition schemes.

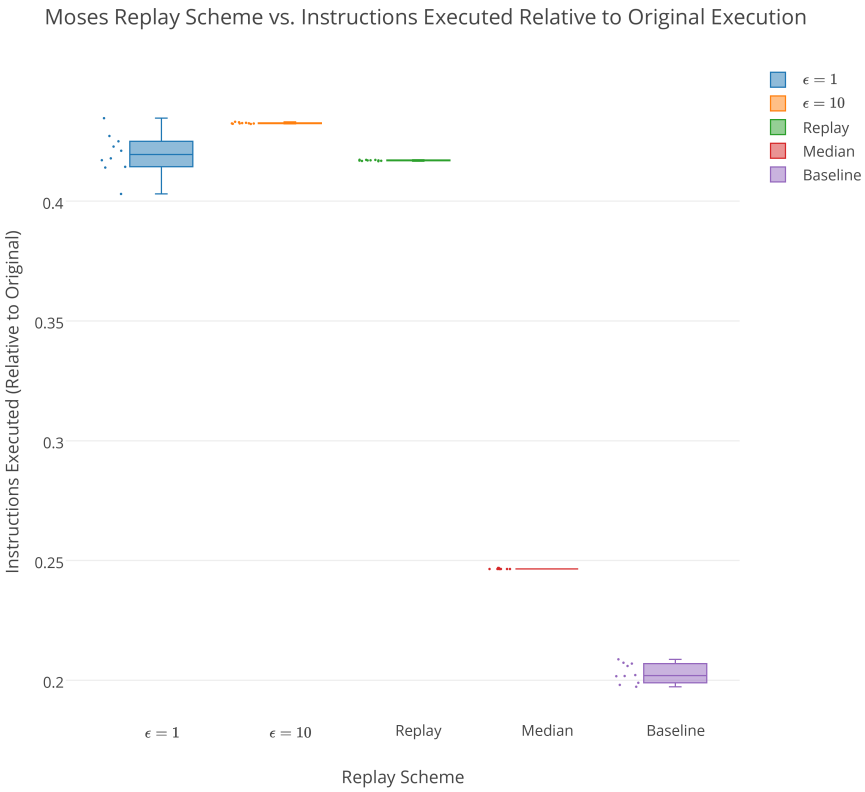

Figure 9: IPC slowdown for model runs for 10 the Moses translation runs on 100 inputs.

overheads) and also be statistically similar for different inputs. This gives a statistical guarantee that any adversary will not be able to recover which was the true input with small probability of error, which can be tuned using a privacy parameter ϵ. It is important to emphasize that if the real program deviates from our assumptions, this only hurts performance but never privacy, since our trace is always generated from the model we design.

Applying our scheme to a shared memory queue which feeds to the memory controller on a Westmere-like out of order (OoO) processor, we were able to greatly decrease the success of intelligent attackers. These adversaries, which had been able to perfectly predict private workloads before noise addition, had their classification accuracy reduced - in some cases to 50%, and at times even worse than randomly guessing. Additionally, we show through simulations in ZSim that our queue modifications for the memory controller only creates overheads around 1.5x for lower values of perturbation to roughly 3x for nearly perfect privacy.

References

[1] Inc. Amazon.com. Amazon ec2, 2016.

[2] Miguel E. Andrés, Nicolás Emilio Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: differential privacy for location-based systems. In Sadeghi et al. [23], pages 901–914.

[3] Microsoft Corporation. Azure, 2016.

[4] Inc. DigitalOcean. Digitalocean, 2016.

[5] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Ding-Zhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008.

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[7] Casen Hunger, Mikhail Kazdagli, Ankit Singh Rawat, Alexandros G. Dimakis, Sriram Vishwanath, and Mohit Tiwari. Understanding contention-based channels and using them for defense. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*, pages 639–650. IEEE Computer Society, 2015.

[8] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recogn.*, 44(9):2231–2240, September 2011.

[9] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[10] Philipp Koehn. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand, 2005. AAMT, AAMT.

[11] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In John A. Carroll, Antal van den Bosch, and Annie Zaenen, editors, *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics, 2007.

[12] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[13] David Levinthal. Performance analysis guide for intel® core™ i7 processor and intel® xeon™ 5500 processors, 2009.

[14] Linux. perf: Linux profiling with performance counters, 2015.

[15] Fangfei Liu, Qian Ge, Yuval Yarom, Frank McKeen, Carlos V. Rozas, Gernot Heiser, and Ruby B. Lee. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016*, pages 406–418. IEEE Computer Society, 2016.

[16] Texas Department of State Health Services. Hospital inpatient discharge public use data file, 2016.

[17] Sewoong Oh and Pramod Viswanath. The composition theorem for differential privacy. Technical report, Technical Report, 2013.

[18] Oracle. Mysql employees sample database, 2016.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In Ahmed K. Elmagarmid and Divyakant Agrawal, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 735–746. ACM, 2010.

[21] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, pages 199–212. ACM, 2009.

[22] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 10(1):16–19, 2011.

[23] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. ACM, 2013.

[24] Daniel Sanchez and Christos Kozyrakis. Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In Avi Mendelson, editor, *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, pages 475–486. ACM, 2013.

[25] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. Avoiding information leakage in the memory controller with fixed service policies. In Milos Prvulovic, editor, *Proceedings of the 48th International Symposium on Microarchitecture, MICRO 2015, Waikiki, HI, USA, December 5-9, 2015*, pages 89–101. ACM, 2015.

[26] Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In Alex Nicolau, Xiaowei Shen, Saman P. Amarasinghe, and Richard W. Vuduc, editors, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013*, pages 135–146. ACM, 2013.

[27] Jakub Szefer and Ruby B. Lee. Architectural support for hypervisor-secure virtualization. In Tim Harris and Michael L. Scott, editors, *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, London, UK, March 3-7, 2012*, pages 437–450. ACM, 2012.

[28] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In Dean M. Tullsen and Brad Calder, editors, *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, pages 494–505. ACM, 2007.

[29] Larry Wasserman and Shuheng Zhou. A statistical framework for differential privacy. *Journal of the American Statistical Association*, 105(489):375–389, 2010.

[30] Qiuyu Xiao, Michael K. Reiter, and Yinqian Zhang. Mitigating storage side channels using statistical privacy mechanisms. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1582–1594. ACM, 2015.

[31] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 305–316, New York, NY, USA, 2012. ACM.

[32] Yinqian Zhang and Michael K. Reiter. Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In Sadeghi et al. [23], pages 827–838.