STOPP-Q: Statistical Trace Obfuscation using Privacy-Preserving Queues — Technical Report

ABSTRACT

In this paper, we study problems if information leakage across *utilization channels*, where applications accessing a resource leak sensitive information to an attacker sharing the same resource. Examples of this include mutually distrustful virtual machines (VMs) that share hardware accelerators or memory controllers or whose hardware utilization is visible to a malicious hypervisor. We introduce STOPP-Q — the first technique that can alter the utilization of shared resources for *provably* quantifiable privacy guarantees. STOPP-Q uses profiling to learn the typical behavior of the application across inputs and shapes the utilization trace at run time to provide statistical bounds on the attacker's ability to map an observed trace to an input. We build this obfuscation mechanism into a *privacy-preserving queue* and use it to share a cryptographic accelerator among mutually distrustful applications on a RISC-V multi-core processor. Simulating on database, graph, and text-translation applications, we find that STOPP-Q can secure the memory queue side-channel by reducing the accuracy of an extensively trained classifier to no better than random chance. Furthermore, STOPP-Q only incurs roughly 1.5–3× instructions per cycle (IPC) slowdown, compared to a baseline IPC slowdown of 2.7–7× for partitioning-based schemes. We thus introduce a fundamental hardware building block that can secure enclaves against utilization channel attacks.

1. INTRODUCTION

Isolation is a fundamental primitive for secure computing that is surprisingly difficult to enforce in practice. For example, Virtual Machines (VMs) in the cloud are isolated from one another and from the underlying hypervisor using *enclaves* [1, 2, 3]. While these enclaves prevent direct access to a VM's memory from a malicious *adversary*, it is still possible for such an adversary to *infer* sensitive information through *side-channels*. This adversary may therefore learn confidential information such as secret keys, database queries, or program inputs [4, 5, 6, 7, 8, 9, 10].

Even when VMs are assigned to separate cores, side-channel attacks may still succeed by observing utilization of shared resources such as the last level cache (LLC) [4, 6, 7], memory controllers [11, 12, 9], and even hardware units such as AES [13] and random number generators [14]. Merely observing a time series of utilization is sufficient to infer secret keys [6] or actively leak data at over 500 Kbps bitrate [13].

One way to close side-channels is to *normalize* a program's observable execution trace, either through static partitioning or forcing worst-case behavior. Normalization is useful for certain attacks that rely on large resources but often causes high performance overheads [11, 12, 15]. For example, partitioning the LLC is an effective way to close side-channels where an adversary VM contends for cache lines to observe a victim VM's cache usage [16, 17]. Prior work has also normalized observations through memory-controllers by time multiplexing [11, 12] or guaranteeing worst-case service [15] to contending VMs. Compile-time solutions [18, 19, 20] have also attempted to normalize the side-channel trace of a program by executing decoy paths. However, normalization is unacceptable when these costs excessively impact program performance.

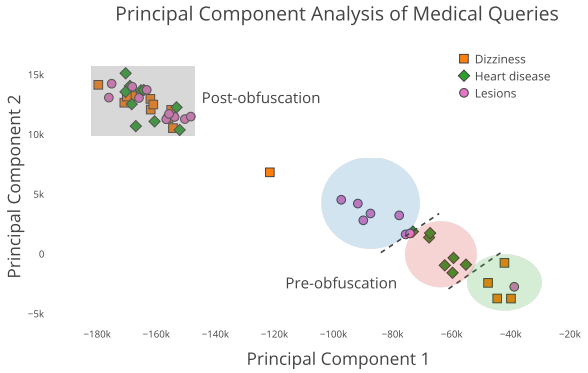

Figure 1: Principal Component Analysis on medical database queries' traces pre- and post-obfuscation. Each point represents one run of a SQL query with a disease category as input. Pre-obfuscation, there is a clear separation among the inputs (diseases), as shown by the dashed lines. Post-obfuscation, the inputs look closer to each other. STOPP-Q quantifies this indistinguishability among inputs and enables users to trade it off for better performance.

An alternative direction to closing side-channels is to *add noise* to or *obfuscate* observable side-channel traces [21, 22, 23]. For example, VAX-VMM introduced "fuzzy" time to add noise to system level clocks that are needed to exploit side- and covert-channels reliably [22]. As an extreme design point, cryptographic techniques have been used to completely randomize the observable program trace [8, 24]. Recent work attempts to protect attacks by *shaping* traffic over the channel to fit a fixed distribution. These noise-based defenses can potentially enable security with low overheads and scale to a large number of contenders. This is especially true if the workload has common-case behavior that is significantly better than the worst case.

Noise-addition based defenses, however, have a major unsolved problem — they cannot provide *provable* information leakage guarantees against an *intelligent* adversary while trading off privacy vs. performance overhead. Fuzzing clocks or hardware timers only perturbs observed timer values within a predetermined 'epoch' [21]. Randomizing cache mappings makes it difficult to contend for a specific cache line, but lets an adversary measure total cache utilization [25]. Additionally, memory traffic shaping in prior work [26, 27] cannot guarantee that (a) fitting the memory trace to a distribution prevents information leakage, and (b) the proposed mechanisms successfully fit the adversary-observable trace to a desired distribution. In summary, while current noise-addition techniques may improve correlation-based metrics [28, 29], they cannot definitively prove that secrets remain confidential.

In this paper, we study the problem where an application runs inside an enclave and makes requests to shared hardware with an adversary observing the entire request trace to try to infer application secrets. This model directly applies to side-channels based on shared memory controllers and hardware accelerators that interface to the rest of the system through a shared request *queue*.

Figure 1 demonstrates an example attack where an adversary observes the memory request trace for three different medical disease queries (that differ in the disease used as input) on the Texas Hospital Discharge Discharge database [30]. We run each query 100 times, generate a trace of median memory requests per time window (of 500k cycles), and perform Principal Component Analysis (PCA) on the resulting time series. Figure 1 plots the top two principal components, and it is clear that the adversary can distinguish the three queries (pre-obfuscation, bottom-right).

Our key conceptual contribution is a new class of side-channel defenses — STOPP-Q — that *makes a program's utilization trace statistically indistinguishable across different inputs*. We assume an intelligent adversary with arbitrary granularity of a shared resource's utilization rate. STOPP-Q models the utilization trace as a time series and closes information leakage across *arbitrary intervals of time observed at any granularity*. STOPP-Q first trains on the time series produced by a program's resource utilization and adds carefully calibrated noise in order to preserve privacy. At runtime, the system shapes the program's request time series to follow this privacy-preserving model. STOPP-Q's noise addition algorithm *guarantees* a lower bound on the indistinguishability between any two inputs for any adversary observing the channel. At the same time, STOPP-Q's noise is generated such that the performance overhead is minimized.

Figure 1 shows the result of applying the STOPP-Q methodology to obfuscate the memory request rate. The three queries' traces overlap (top-left) — the results of this PCA give an intuition for how our obfuscation methodology shapes program behavior of differing inputs to be increasingly similar to one another. In Section 6.1, we additionally evaluate our defenses against a more intelligent adversary.

We build on the key idea of adding noise to utilization time series to make the following specific contributions:
1. Security model based on input privacy. STOPP-Q adds noise to resource utilization traces to ensure that an adversary cannot distinguish among the program's inputs even if it can observe *every* access by the victim to the resource. To do so, STOPP-Q proposes a *median* replay technique that *perfectly* anonymizes each program trace, and an *optimal* scheme that minimizes slowdown within *provably* configurable privacy bounds.
2. Architectures using privacy-preserving queues. We propose a *secure queue* that adds noise to observable utilization traces which designers may use to build hardware that is shared among trust domains but is expensive to partition. For example, STOPP-Q can statistically multiplex hardware such as the uncore and accelerators.
3. Security evaluation against intelligent adversary. We evaluate STOPP-Q's security guarantees with different *privacy parameters* against a classifier that is extensively trained on the output traces and can perfectly determine inputs to an application pre-obfuscation. We demonstrate that it fares no better than random guessing after applying STOPP-Q.
4. Performance evaluation with real-world applications. We simulate a cloud server system running realistic applications (e.g. medical database, graph processing algorithms) where several cores send memory requests through our secure queue. We show that STOPP-Q incurs only 1.5–3× IPC slowdown compared to a baseline time-partitioning IPC slowdown of 2.7–7×.
5. Hardware evaluation of a shared accelerator. We demonstrate STOPP-Q's secure queue through a synthesizable hardware design where two RISC-V core tiles [31, 32] share a cryptographic hash accelerator using our secure queue. The implementation on an FPGA board requires only a 0.4% increase in Slice LUTs and 1 additional Block RAM, with no impact on the cores' critical path.

In summary, STOPP-Q shows — for the first time — a defense that maintains provable security guarantees and incurs a low overhead when adding noise to observable traces. Before we dive into the details of STOPP-Q's hardware design in Section 3 and security guarantees in Section 4, we discuss utilization-based side-channel attacks and existing defenses against them.

2. BACKGROUND

2.1 Threat Model

Figure 2 shows STOPP-Q's threat model. We assume three principals: *tenants* who rent cloud infrastructure and run their applications in VMs, *cloud providers*, and *hardware manufacturers*. The tenants are mutually distrustful and one tenant VM should not leak secret data to another tenant's VM. Malicious tenants will attempt to become co-resident with a victim VM [4] and then launch side-channel attacks through the shared hardware. We consider the cloud provider to be un-

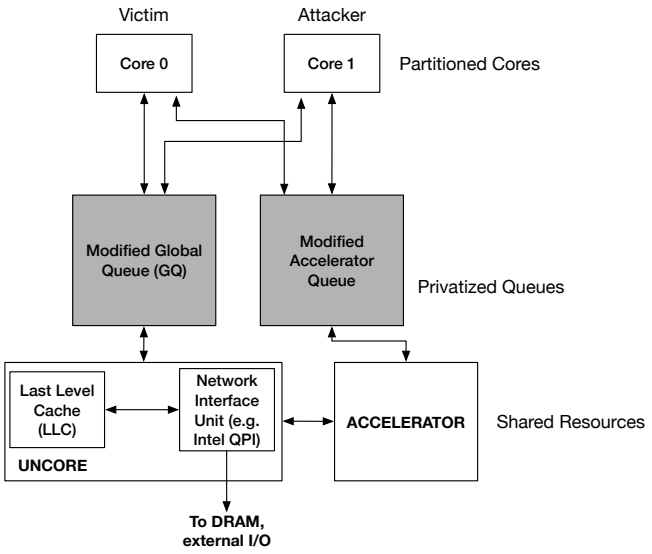

Figure 2: Threat model. A victim application uses resources such as the uncore for memory requests or hardware accelerators. Attackers such as a co-resident process or a hypervisor can observe the victim's resource utilization and infer its secrets. STOPP-Q's privacy-preserving queue obfuscates such utilization channels.

trusted. Hence, VMs run confidential computation inside hardware-based enclaves [1, 2] so that hypervisors cannot observe their secret data. Recent work shows that extended enclaves can seal hypervisor-level side-channels [33], leaving hardware-level side-channels as the critical unsolved problem in enclave-based systems. We assume hardware manufacturers are trusted and also assume an enclave-based system that runs realistic applications per-core as our baseline [34, 33].

Hardware-level side-channels fall into two categories. *Contention* channels where an adversary infers secret data by contending with the victim VM for LLC sets [7, 16], memory controller [11, 15, 12], or hardware accelerators [13, 14], etc. Alternatively, *observation* channels allow an adversary to infer secrets based on the victim's hardware utilization trace (e.g., hypervisor can silently observe performance counters [13] or a malicious memory module can observe the physical address trace [8, 24]).

STOPP-Q's techniques apply to all inter-core contention channels and all software-visible observation channels. Hence, the LLC, memory controller, hardware accelerator, and performance counter channels are within scope while the physical memory address trace is not. We rely on techniques such as oblivious RAM [8, 24] or an encrypted channel to a Hybrid Memory Cube-like physical memory to protect the address bus. Further, we rely on cache partitioning (e.g., using Intel CAT [17] or partition-locked caches [25]) to protect *access-driven* side-channels through the LLC. Partitioning large resources like the LLC has been shown to benefit security and quality-of-service properties with small overheads [35]. Partitioning, however, is an expensive solution to prevent contention through the *queues* — such as the 'Uncore Global Queue' in Intel systems — that lead to *timing-driven* attacks through the shared LLC and memory bandwidth as well as hardware accelerators among multiple cores. In contrast, STOPP-Q's approach of shaping the resource utilization trace of an application does not require static resource partitioning — yet it obfuscates the secret inputs' effect on the contention and performance counter values seen by malicious VMs and the hypervisor.

2.2 Related Work

Side-channel defenses include proactive ones that mitigate attacks and reactive defenses that aim to detect attacks. STOPP-Q is a proactive defense against utilization driven side-channel attacks.

Proactive defenses. In addition to the partitioning-based defenses mentioned before, several proactive techniques add noise to or shape the utilization of shared resources over time. For example, Düppel [36], presented by Zhang et al., provides a mechanism for periodically cleansing the L1 cache to mitigate side-channels on time-shared caches. Similary, TimeWarp [21] adds noise to hardware timer's (`rdtsc` instruction's) output to mitigate fine-grained timing attacks. In a forthcoming paper[1], Zhou et al. [27] adapt memory traffic shaping for quality of service [26] to instead mitigate side channel leaks through the memory controller. We observe that none of the above noise-addition schemes provide provable guarantees for information leakage. We discuss fundamental limitations of Zhou et al.'s scheme (Camouflage [27]) for memory traffic shaping in the following sub-section — while Camouflage is close to STOPP-Q in terms of its intention, it leaves open high-bandwidth long- and short-term timing channels and does not offer provable bounds on information leakage.

Reactive defenses. A complementary line of work attempts to *detect* attacks when they occur instead of adding defenses into the potential victim up front. Such defenses include hardware- or instruction-level malware detectors that monitor micro-architectural signals or performance counters and look for signatures [37, 38, 39] of malicious execution, contention [40, 13], or anomalies [41, 42, 39]. Such techniques apply machine learning models to yield variable true- and false-positive rates, but (like proactive defenses above) do not formally guarantee confidentiality. Such *hardware- based malware detectors* are complementary to STOPP-Q and can detect low-level attacks against system integrity, such as RowHammer [43].

Differentially private mechanisms. STOPP-Q's goal of preserving inputs' privacy is inspired by work on adding noise to make database query outputs *differentially private* (e.g., [44, 45, 46, 47, 48, 49, 50]). Differential privacy essentially requires that the probabilities that a query produces a given output are close (to a configurable degree based on a privacy budget) when

[1]We received a preliminary copy dated November 9, 2016

the queries are run on two databases which differ in at most one row. Differential privacy usually requires measuring how sensitive a query's output is to a change in input and adds an appropriate amount of noise based on the query's sensitivity and the privacy budget.

Prior work attempts to determine the optimal noise under the traditional differential privacy mechanisms, which do not easily apply in our setting [51, 52, 49, 53]. Unlike STOPP-Q, these setting also does not require adding noise to and then shaping time series of utilization traces. STOPP-Q requires adapting these metrics to time series data where each row is instead a time series labeled with an input. Further, the output of the mechanism must be shaped in real-time as it is being created by the program. In contrast, prior work on differentially private time series [54] only handles aggregate queries on time series data (e.g., positional queries on a GPS trace) without an online component.

2.3 Traffic Shaping

Recent work by Zhou et al. introduces Camouflage [27], a memory traffic-shaping design that extends MITTS [26] to mitigate timing channel attacks through memory controllers. However, we demonstrate that Camouflage cannot fundamentally bound the amount of information leaked through the memory request time series, and we describe an example covert channel that breaks Camouflage at bandwidths on the order of 10 Kbps.

Camouflage relies on mechanisms proposed in MITTS to shape the memory traffic at the granularity of *replenishment windows*. Each replenishment window, Camouflage initializes a histogram which bins the latency between consecutive memory requests The value of each bin is the number of `credits`, or requests, that can be issued for each inter-request latency. Camouflage then *drains* the credits in the histogram during each replenishment window by stalling requests until the latency fits a bin with available credits. If credits are underutilized, they roll over into subsequent replenishment windows where they are used to generate fake requests (to random addresses) that have lower priority than actual requests. Camouflage's goal is to shape the observable utilization to fit the histogram's distribution, and prevent leakage when the attacker's observations are on "longer time scales" than the replenishment window.

Camouflage's fundamental limitations. However, a significant limitation of Camouflage is ignoring dependencies among memory requests across replenishment windows. The random variable X in Camouflage, which corresponds to actual inter-arrival times obtained from program traces, has strong time dependency in practice: memory traces are well-known to have bursts and hence small inter-arrival times likely follow other small inter-arrival times. This leads to two major concerns:

(1) Preventing information leakage. Camouflage *drains* the histogram of request-credits and allows credits from one replenishment window to affect an arbitrary number of following windows – this allows time dependencies from $X(t)$ to transfer to $Y(t)$ (the shaped traffic). An attacker can thus sense the effect of draining by measuring the latency of its own memory requests. Specifically, if the victim makes no memory requests during two consecutive replenishment windows, $Y(t)$ for the second window is generated using only the roll-over credits from previous window and thus creates low contention with a spy process. As a result, even a weak attacker who can only sense 'long-term' effects over several windows can still measure the difference between a victim who makes no real memory accesses from one that drains its credits during each window. In practice, an attacker can create a covert channel that transmits one bit every two replenishment windows. Assuming replenishment windows of 1,000 memory cycles [27] and a memory clock speed around 200–300 MHz, sending 1 bit per 2000 cycles can create channels with bandwidth of 10s of Kbps. The details of this covert channel are described in [55].

(2) Quantifying information leakage. Further, Camouflage fundamentally leaks information in ways that are very hard to test or control for in an application. A sequence of time-dependent random variables is called a *stochastic process*. Therefore, Camouflage's actual problem consists of estimating entropy and mutual information of two stochastic processes $X(t), Y(t)$. This is a significantly more challenging problem compared to estimating the entropy and mutual information of two *scalar* random variables X, Y from independent samples (which is how Camouflage estimates information leakage) [27, 56, 57, 58].

Without making any assumptions on the type of memory in the stochastic process $X(t)$, estimating entropy requires computing histograms of all possible combinations of t values, which scales exponentially in the number of observed timeslots [56, 57, 58]. Unfortunately, learning the time-scales for which these time-dependencies exist in request time series is very hard. Worse, a covert channel can *deliberately* create such time dependencies to leak data at high bandwidths.

Shaping in STOPP-Q. In this paper, we explicitly handle time dependencies and prove precise bounds on the amount of leaked information *even if the adversary has perfectly precise timing information at any resolution*. The key reason (explained in further detail in Sections 3 and 4) is the following: our approach learns a distribution for each time interval t in a program's execution, and then *samples* $Y(t)$ from this distribution. This allows us to generate a masked trace $Y(t)$ that is *independent across time* (i.e., is secure), but not identically distributed (and hence improves performance by varying memory requests based on program phases determined using offline analysis). By learning the statistics of $X(t)$ for each time t we ensure that our samples are quite close to the real trace, but privacy can be controlled by exploiting independence across time for the process $Y(t)$ that the adversary observes.

3. STOPP-Q SYSTEM DESIGN

As shown in Figure 2, STOPP-Q introduces a *privacy-preserving queue*, replacing existing ones that interface with shared hardware resources such as a memory con-

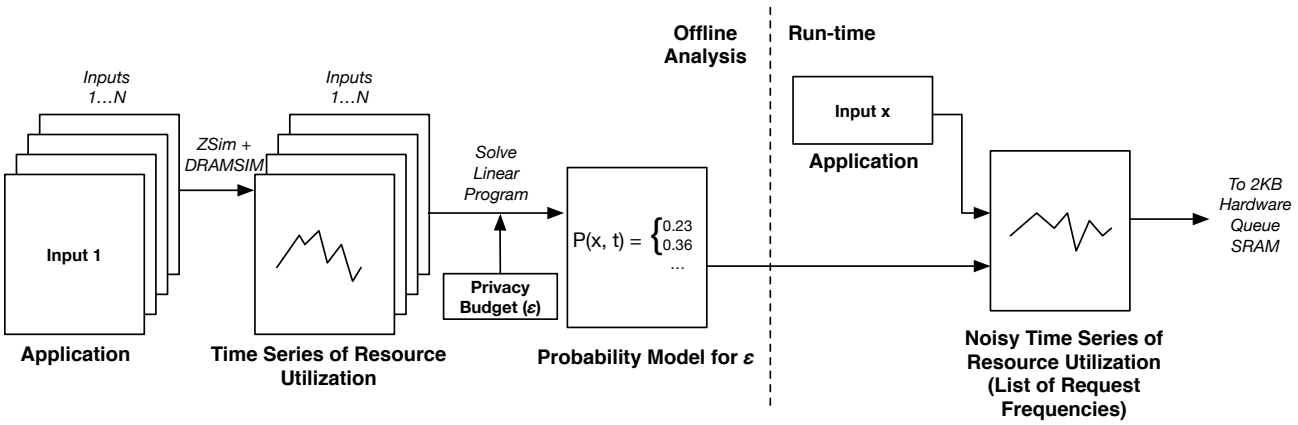

Figure 3: STOPP-Q system flow. *Offline*: We create a model of utilization time series across all inputs, and add minimal noise to the model that enforces privacy constraints. *Run-time*: STOPP-Q generates a time series by sampling from the noisy model, and uses it to shape the current execution trace.

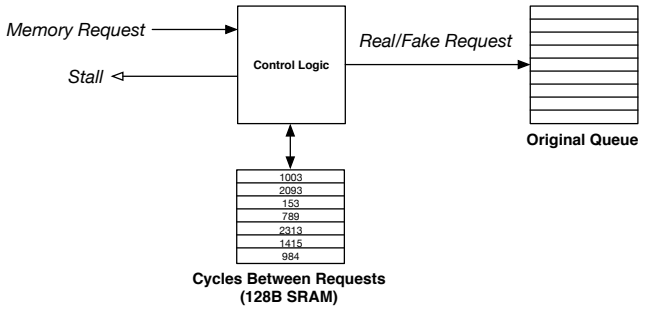

Figure 4: STOPP-Q's hardware queue. The noisy time series of request frequencies in Figure 3 is loaded into a small SRAM; the control logic ensures real/fake requests are issued according to the time series.

troller or an accelerator. The privacy-preserving queue shapes a program's utilization of the shared resource to be statistically indistinguishable across different inputs with minimal overhead. STOPP-Q has an *offline* analysis component which creates a time series model of how a program uses the shared resource across several inputs. For example, each point in the time series for the memory request queue is the number of requests sent in a time window (e.g. 100,000 cycles). The offline analysis then adds noise to the model and generates a new, privatized time series which is loaded into the queue's SRAM. At run-time, STOPP-Q's hardware logic shapes the resource utilization to match the noisy time series model loaded from SRAM. We now walk through the offline and online components of STOPP-Q (Figure 3), followed by the hardware the queue uses to shape traffic (Figure 4).

Modeling utilization traces as a time series. As shown in Figure 3, a user runs a program with different inputs (e.g., queries to a database) and records a time series of resource utilization using architectural simulators or by reading hardware counter values for each input. Each value in the time series corresponds to the number of requests within one *time window* of execution. We note that the size of a time window is a purely performance-related parameter that a user can choose. STOPP-Q's privacy guarantees are independent of the time window and apply to adversaries who observe utilization at any time resolution. STOPP-Q runs *each* input many times (100 in our experiments) to learn the distribution of request rates for each input in *each time window* — this characterizes the inherent variation in the time series due to non-determinism in the underlying micro-architecture.

Adding noise to request time series. Next, the time series for all program inputs and iterations are used to generate a probability model. We begin with a *median* technique that yields a perfectly private trace and then extend it to enable a privacy-performance tradeoff. Our first observation is that *within* one time-window, shaping the time series of resource utilization to have a *fixed* request rate leaks *no information*, regardless of input or program behavior. To extend this to a time series, for each time window t, we can use the *median* of the request rates observed for window t across all inputs — this yields the best performance for the constant request per window scheme. At the same time, since the output request rate is the same for all inputs, the attacker learns no secrets. We find that the median scheme typically performs better than partitioning-based schemes, achieving over 20% more instructions per cycle (see Section 6). However, the median scheme does not allow trading off privacy for even better performance.

To provide a tunable security vs. performance tradeoff, we introduce an *optimal* obfuscation scheme to shape

the observed request rate to a non-deterministic value in each time window. Within each window, STOPP-Q enforces a constant latency between requests. As a result, STOPP-Q closes **long-term** leaks across time windows (using the noisy distribution), and **short-term** leaks within a time window using a constant rate within the window. However, instead of choosing the median for each time window, we sample the request rate from an input-specific distribution that is crafted to make different inputs statistically indistinguishable with a bounded probability while minimizing the performance loss. The user supplies a privacy parameter ϵ that bounds the probability of an adversary distinguishing the specific time window are across different inputs (refer to Section 4.2 for the cost model evaluation). The privacy parameter ϵ is configurable; while prior work has discussed how to pick ϵ effectively [59, 60], we empirically evaluate the choice of ϵ from both a security perspective and a performance perspective (see Section 6).

The resulting probability model is used at run-time to generate a noisy time series for the specific input being used. This obfuscated time series is loaded into memory to be used by our hardware support described next.

STOPP-Q hardware support. STOPP-Q's hardware is easy to integrate into the overall system and its own hardware requirements are small. STOPP-Q replaces queues in front of shared hardware with privacy-preserving versions. While the offline model generation and noise addition require a rigorous analysis described in the next section, the run-time component only requires SRAM to store the privacy-preserving time series and simple logic to maintain a constant rate of utilization within a window (by issuing either actual or fake requests).

4. MATHEMATICAL FRAMEWORK

In this section, we provide a formal mathematical framework for our noise addition scheme. We state the privacy constraints our scheme has to satisfy and explain how we compute the optimal noise to add under those constraints.

For notational simplicity, we consider outputs from the first time window throughout most of the section. Recall that in Section 3, we model the utilization trace as a time series, where we consider *time windows* of user-specified length in cycles, and a point in the time series represents the number of requests sent in a time window. We discuss how to generalize outputs to time series (e.g., corresponding to the multiple time windows over which a program execution spans) at the end of this section in Section 4.5.

4.1 Notation

Consider a program that takes m types of inputs labeled from 1 to m. We denote the unperturbed output from input i as the random variable X_i, and the output for input i with noise added as the random variable Y_i. X_i is sampled from the *intrinsic distribution* x_i over a set of non-negative real numbers $\mathcal{X}$, and takes on the value $k \in \mathcal{X}$ with probability $x_i(k)$ (i.e., $x_i(k) = \Pr[X_i = k]$). Similarly, Y_i is sampled from the *target distribution* y_i over a set of non-negative real numbers $\mathcal{Y}$, and takes on the value $k' \in Y$ with probability $y_i(k')$ (i.e., $y_i(k') = \Pr[Y_i = k']$).

For example, in our PostgreSQL medical database experiments in Section 6, inputs correspond to the queries we make on the database, and we have $m = 50$ different queries. Referring to Figure 4, the unperturbed output X_i corresponds to the number of memory requests PostgreSQL would send to the Control Logic on query i in the first $500,000$ cycles (the time window size). This number of sent requests follows the distribution x_i, which we may learn by running each query multiple times and observing the number of memory requests sent. We emphasize that if this learning is approximate, this will only lead to worse performance but will not affect privacy. The perturbed output Y_i is the total number of real and fake memory requests that get sent out to the queue (see Figure 4), and y_i is the target distribution that we want Y_i to conform to. $\mathcal{X}$ and $\mathcal{Y}$ would then correspond to the number of memory requests that may be made during a time window.

Given the intrinsic distributions x_i for all $i \in [m]$, our goal is to compute the target distributions y_i that are close to each other (with the notion of closeness defined in Section 4.2) while minimizing the performance loss (under the cost model defined in Section 4.3). This ensures that an adversary observing the perturbed output Y_i cannot learn much about which input was fed into the program, since the output would look similar regardless of which input was actually fed into the program.

4.2 Differential Privacy

We adapt the concept of differential privacy [45] to quantify the closeness of the target distributions. Specifically, we want to enforce our program to only leak 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, when we say we want target distributions to be close, we mean that we want a (tunable) ϵ level of privacy defined as follows:

DEFINITION 1. *A noise addition scheme offers an ϵ level of privacy if for any two inputs $i, j \in [m]$, their target distributions are within a factor of e^ϵ for all possible output values:*

$$y_i(k) \leq e^\epsilon y_j(k), \qquad \forall i, j \in [m], \forall k \in \mathcal{Y}.$$

The privacy parameter ϵ dictates how much freedom we have in choosing the target distributions to ensure the desired level of privacy is not violated. Smaller values for ϵ require target distributions to be closer to each other, making them less distinguishable and thus give higher privacy guarantees. For example, when $\epsilon = 0$, the inequalities in Definition 1 are satisfied only when $y_i(k) = y_j(k)$ for all i, j and k, meaning the target distributions for all inputs have to be exactly the same. Larger values for ϵ allows target distributions to be further apart and we have more freedom in how we perturb the outputs. This additional amount of freedom

in picking the target distributions also allow us to incur a lower performance loss (our cost model will be stated in the next subsection). 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 outputs caused by different inputs [61, 62].

4.3 Cost Model

We incur a performance loss when the values of the original output X_i and perturbed output Y_i do not match (for example, due to having to stall memory requests or introduce fake memory requests so that our request frequency conform to the perturbed output). The larger the discrepancy between X_i and Y_i, the higher the performance loss. Therefore, we define the cost of a noise addition scheme to be the mean absolute difference between the original and perturbed output values.

DEFINITION 2. *The cost incurred by a noise addition scheme under the* absolute-difference cost model *is given by*

$$\sum_{i\in[m]} \mathbf{E}\left[|X_i - Y_i|\right] = \sum_{i\in[m]} \sum_{k'\in\mathcal{Y}} y_i(k') \left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'| \right).$$

In the scenarios we consider, inputs are drawn uniformly at random (i.e., input i is fed into the program with probability $1/m$, for all $i \in [m]$). When inputs are not drawn uniformly at random, we can simply weight the contribution of each input to the cost above accordingly.

4.4 Optimizing Noise Addition

Given the set of intrinsic distributions $\{x_1, x_2, \ldots, x_m\}$, the privacy constraints from Definition 1, and the cost model from Definition 2, we seek to find the set of optimal target distributions $\{y_1, y_2, \ldots, y_m\}$ (minimizing cost) that satisfy all the privacy constraints. This cost minimization problem can be written as the following linear program (LP) in the variables $y_1, y_2, \ldots, y_m$:

$$\text{minimize} \quad \sum_{i\in[m]} \sum_{k'\in\mathcal{Y}} y_i(k') \left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'| \right) \quad (1)$$

$$\text{subject to} \quad y_i(k) \le \mathrm{e}^{\epsilon} y_j(k), \forall i,j \in [m], \forall k \in \mathcal{Y}, \quad (2)$$

$$\sum_{k\in\mathcal{Y}} y_i(k) = 1, \forall i \in [m], \quad (3)$$

$$y_i(k) \ge 0, \forall i \in [m], \forall k \in \mathcal{Y}. \quad (4)$$

The objective function (1) is simply the cost from Definition 2. The constraints (2) restrict the target distributions to lie in the polytope bounded by the privacy constraints (from Definition 1) for this given ϵ, while constraints (3) and (4) simply restrict the solutions to be probability mass functions. Together, these constraints define a *privacy polytope* in which the solution to the LP lie. The solution to this LP are the optimal target distributions $y_1, y_2, \ldots, y_m$. At runtime, when input i is fed to the program, our noise addition algorithm would simply output a value sampled from y_i.

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. 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 (points with non-zero probabilities) of the target distributions $\{y_i\}_{i\in[m]}$ is the union of the supports of the intrinsic distributions $x_1, x_2, \ldots, x_m$, given that the supports of the intrinsic distributions are in the set of possible outcomes $\mathcal{Y}$ of the target distributions.*

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 $y^* = \{y_i^*\}$ in which $y_i^*(k^*) > 0$ for some i, where $k^* \notin \bigcup_i \mathrm{supp}(x_i)$ (we denote the supports of a distribution x_i as $\mathrm{supp}(x_i)$). We construct another solution $\hat{y} = \{\hat{y}_i\}$ as follows:

1. Pick any $k^* \in \bigcup_i supp(y_i^*) \setminus \bigcup_i supp(x_i)$.
2. If $\sum_i y_i^*(k^*) \sum_{k>k^*} x_i(k) \ge \sum_i y_i^*(k^*) \sum_{k<k^*} x_i(k)$, we set $\hat{k} = \min\{k : k \in \bigcup_i supp(x_i) \text{ and } k > k^*\}$. Otherwise, set $\hat{k} = \max\{k : k \in \bigcup_i supp(x_i) \text{ and } k < k^*\}$. Intuitively, $\hat{k}$ is either the smallest support of $\{x_i\}$ that is larger than k^* or the largest support that is smaller than k^*. It is chosen in a way that minimizes the objective function when we move all the mass from $y_i^*(k^*)$ to $y_i^*(\hat{k})$.
3. Construct $\hat{y}$ from y^* as follows. For all i,

$$\hat{y}_i(k) = \begin{cases} y_i^*(\hat{k}) + y_i^*(k^*) & \text{if } k = \hat{k}, \\ 0 & \text{if } k = k^*, \\ y_i^*(k) & \text{otherwise.} \end{cases} \quad (5)$$

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

LEMMA 4. *$\hat{y}$ 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 $y_i^*(k^*)$ to $y_i^*(k')$. To see why constraints (2) are still satisfied, consider the following cases.

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

$$\begin{aligned} \hat{y}_i(k) &= y_i^*(\hat{k}) + y_i^*(k^*) \\ &\le \mathrm{e}^{\epsilon} y_j^*(\hat{k}) + \mathrm{e}^{\epsilon} y_j^*(k^*) \\ &= \mathrm{e}^{\epsilon} \hat{y}_j(k). \end{aligned}$$

Case 2. $[k = k^*]$ Since $\hat{y}_i(k) = 0$, the constraints are trivially satisfied.

Case 3. $[k \notin \{\hat{k}, k^*\}]$ Since $\hat{y}_i(k) = y_i^*(k)$, the constraints are still satisfied. □

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

$$\sum_{i\in[m],k'\in\mathcal{Y}} \hat{y}_i(k')\left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'|\right) - \sum_{i\in[m],k'\in\mathcal{Y}} y_i^*(k')\left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'|\right) \leq 0.$$

PROOF OF LEMMA 5.

$$\begin{aligned}
&\sum_{i\in[m],k'\in\mathcal{Y}} \hat{y}_i(k')\left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'|\right) - \sum_{i\in[m],k'\in\mathcal{Y}} y_i^*(k')\left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'|\right)\\
&= \sum_{i\in[m],k'\in\mathcal{Y}} (\hat{y}_i(k')-y_i^*(k'))\left(\sum_{k\in\mathcal{X}} x_i(k)|k-k'|\right)\\
&= \sum_{i\in[m]} y_i^*(k^*)\left[\left(\sum_{k\in\mathcal{X}} x_i(k)|k-\hat{k}|\right)-\left(\sum_{k\in\mathcal{X}} x_i(k)|k-k^*|\right)\right]\\
&\qquad\qquad \text{(Equation (5) in Step 3)}\\
&\leq 0\\
&\qquad\qquad \text{(Step 2). } \square
\end{aligned}$$

The following lemma follows from above.

LEMMA 6. $\hat{y}$ *is an optimal solution to the LP.*

PROOF OF LEMMA 6. By Lemma 4, $\hat{y}$ will not violate constraints (2)-(4) of the LP. By Lemma 5, $\hat{y}$ still minimizes the objective function. Therefore, $\hat{y}$ is an optimal solution to the LP. $\square$

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

PROOF OF PROPOSITION 3. Given an optimal solution $y^* = \{y_i^*\}$ in which $y_i^*(k^*) > 0$ for some i, where $y^* \notin \bigcup_i \text{supp}(x_i)$, we can always construct another optimal solution $\hat{y}$ that has the desired property, using the above algorithm with the following additional step.

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

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

This proposition allows us to reduce the size of the problem since we only need to consider supports of the intrinsic distributions to compute the optimal target distributions.

If the intrinsic distributions corresponding to each input are deterministic (i.e., for any input i, $x_i(k) = 1$ if $k = \alpha_i$ for some value $\alpha_i \in \mathcal{X}$, and $x_i(k) = 0$ otherwise), then the following corollary follows from the previous proposition:

COROLLARY 7. *Suppose* $x_i(\alpha_i) = 1$ *where* $\alpha_i \in \mathcal{X}$ *for all input* $i \in [m]$ *(i.e., the intrinsic distributions are deterministic). Then there is at least one optimal solution such that the union of the supports of the target distributions* $y_1, y_2, \ldots, y_m$ *is a subset of* $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, *given that* $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ *is a subset of the set of possible outcomes* $\mathcal{Y}$ *of the target distributions as well.*

When the intrinsic distributions are deterministic, the above properties help in reducing the computational complexity by drastically reducing the size of our LP, since usually $m \ll |\mathcal{Y}|$ (e.g., number of input types v.s. number of possible memory accesses within a time window). Moreover, if $\mathcal{Y}$ is not a finite set (e.g., $\mathcal{Y}$ is the set of all integers), we may not be able to solve the original problem, but we can solve this new LP problem which considers outputs only from the set $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$.

The above properties also help reduce storage requirements when storing the solutions: instead of storing the entire output set $\mathcal{Y}$ and distributions $\{y_i\}$ (requiring $(m+1)|\mathcal{Y}|$ numbers in the naïve way), we need to store only $\{\alpha_i\}$ and $y_i(k)$ for $i \in [m]$ and $k \in \{\alpha_i\}$ (requiring less than $m(m+1)$ numbers), which can be very significant when $m \ll |\mathcal{Y}|$.

To leverage the above savings for outputs that are not deterministic, one heuristic that seems to work well is to first approximate each intrinsic 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. We stress that even if the solution we get is a poor approximate of the original optimal solution, this will only hurt performance and not privacy since the approximate solution would still have to satisfy all the privacy constraints.

4.5 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 as independent across time and apply the previous solution in each step independently. This naïve approach would require computing and storing approximately $m(m+1)T$ parameters which would be prohibitively complex when we have T time steps.

We instead use a formulation that offers better performance and models time-dependencies. Denote the median of x_i at time t as $\alpha_i(t)$ 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 compress this time series by approximating α_i with an autoregressive model with a small order c. This way, we can compress the T points of α_i into $c+1$ coefficients and the c starting points $(\alpha_i(1), \alpha_i(2), \ldots, \alpha_i(c))$, at the cost of additional computations needed to generate predictions for α_i at runtime.

Note that all the different ways of modeling and compressing the time series will never violate the privacy constraints since we are always constraining our target distributions to lie within the privacy 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.

5. MODELING INTELLIGENT ATTACKER

In order to empirically evaluate our proposed privacy mechanism at different privacy parameters, we introduce an *intelligent adversary* which attempts to classify the observed request time series into a set of known inputs. This adversary is capable of obtaining fine-grain timing information and building an accurate model of program behavior over time for each input.

To model such an adversary, we begin with a large and diverse set of feature extraction techniques (including several distance metrics in both time and frequency domains). The adversary then trains a ensemble classifier that uses gradient boosting to assemble weak classifiers such as decision trees into a robust classifier. We present the impact of adding noise on the classification accuracy of this adversary in Section 6.1.

Specifically, we build *feature vectors* from each time series with pairwise distance information between each time series, coefficients from discrete wavelet decomposition, and coefficients from an autoregressive model. We use Dynamic Time Warping (DTW) to capture the distance between time series and each training time series. DTW is used because it provides a more intuitive notion of "closeness" between two time series [63]. 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. Next, we collect the first 20 wavelet coefficients for the discrete transform using the Daubechies 1 (db1) wavelet [64]. 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 the Fourier transform. Lastly, we capture 100 coefficients of an autoregressive model of the time series.

The task of mapping an observed trace to an input can be considered a one-vs-all classification problem — i.e., the classifier selects the input which has the highest probability of producing the observed time series. We report results using a gradient boosting tree regressor [65] trained on the above feature vectors (since it performed better than multi-class SVM, logistic regression, and random forest classifiers).

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 [66] and Moses [67].

We map the one-vs-all classification to pairwise comparisons using the decision trees. Other classification schemes (and thus a differently modeled adversary) may provide slightly different results, but the general notion is that there exists an observable difference between inputs than an adversary can determine and exploit. When building the classifier, we additionally use 5-fold cross validation to reduce the chance of overfitting error as well as the probability of fitting to any noise within the training data.

To evaluate our defense in the worst-case scenario, we assume in our attack simulations that the adversary is able to extract the *exact* resource utilization. To determine the overall efficacy of our modeled adversary, we test adversaries which have been trained on the original time series, the median of a workload's input time series, and noisy time series replayed for different values of epsilons. This test demonstrates a baseline for how accurately a reasonably well-trained adversary with the ability to effectively measure resource utilization would fare against our noise scheme. We note that this is not a definitive measure of how well a different adversary will perform against the different replay and noise schemes, but it nevertheless gives a realistic adversarial model that we can augment (in future work) with metrics that use correlation between time series [28, 29].

6. SIMULATION RESULTS

We now evaluate STOPP-Q's security with different privacy parameters against an intelligent attacker and measure performance improvements over a baseline partitioned system. We implement a memory queue which executes STOPP-Q's methodology in a simulator based on ZSim [68] and DRAMSim2 [69], and evaluate its security-performance trade-offs for three applications

— PostgreSQL with a medical database, text translation using Moses, and graph algorithms on large-scale datasets using Ligra. We use these applications instead of SPEC workloads since they are more representative of sensitive applications with secret data.

To evaluate security, we use the classifier described in Section 5 to measure how effectively our scheme obfuscates the attacker's observed memory request behavior. We simulate queries to completion on a hospital inpatient database [30] using PostgreSQL.

For performance, we simulate a relevant portion of the execution of our benchmarks and report IPC slowdown. We simulate 1 billion cycles of 8 graph algorithms on 10 real-world graphs from the SNAP datasets [70] shown in Table 3, utilizing the Ligra graph processing framework [66] in single-threaded mode for our graph simulations. We also simulate a real-time translation system Moses [67] using a pre-built English-Spanish translation model based on the Europarl [71] language corpus translating 50 sentences from Don Quixote.

For both evaluation criteria, we simulate a set of representative program inputs and begin simulation at the region-of-interest (ROI), such as the start of the PostgreSQL database query. We run each simulation for 100 iterations when training the model and 10 iterations when simulating the noise addition scheme.

6.1 Security Evaluation

We first modified the ZSim [68] memory controller that feeds requests to DRAMSim2 [69] to record the number of memory requests every *time window* (200,000–500,000 cycles in our experiments). This enables us to model an adversary observing 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 individual 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 *multi-class classification* problem.

To test STOPP-Q against the classifier, we simulate an 800 MB PostgreSQL database containing inpatient medical records in a hospital. We find that similar trends hold for the graph and translation experiments as well. 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 5 and 6 show the ROC curve results of our classifier for a memory request trace with an time window of 500,000 cycles before and after our noise addition scheme (for ϵ of 0.1). The ROC curve is calculated for the multi-class 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 nearly perfect accuracy, while the probability of false positives significantly increases when noise is added.

Input	Nodes	Edges
Skitter Internet Topology	1,696,415	11,095,298
Patent Citations	3,774,668	16,518,948
Friendster Social Network	65,608,366	1,806,067,135
Orkut Social Network	3,072,441	117,185,083
Road Network California	1,965,206	2,766,607
Road Network Pennsylvania	1,088,092	1,541,898
Road Network Texas	1,379,917	1,921,660
Live Journal Social Network	3,997,962	34,681,189
Pokec Social Network	1,632,803	30,622,564
Wikipedia Talk Network	2,394,385	5,021,410

Graph Algorithm
Betweenness Centrality
Radii (Graph Eccentricity Estimation)
Connected Components
Triangle Counting
K-Core Decomposition
Breadth-First-Search
Maximal Independent Set
PageRank

Table 3: Graph inputs and algorithms used for Ligra [66] experiments.

6.2 Performance Evaluation

We evaluate the performance costs of our system modifications and demonstrate an IPC overhead of 1.5–3x compared to an IPC overhead of 2.7–7x for a partitioning scheme. We test the IPC slowdown incurred for each of our three benchmarks: PostgreSQL medical db, Ligra [66] (8 different graph algorithms), and the Moses [67] 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 7. 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 time window 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 instead add statistical noise.

Next, we determine the overhead incurred for privatizing inputs to Moses [67] and graph the results in Figure 8. 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

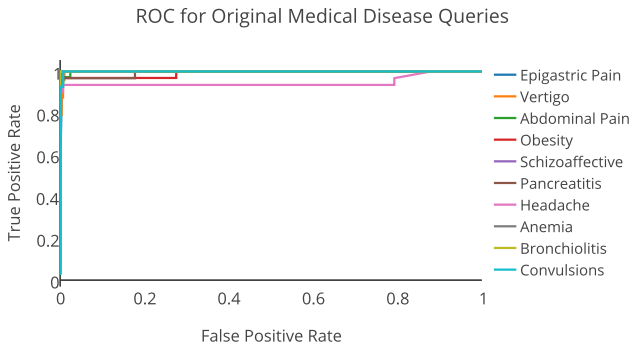

Figure 5: (Pre-obfuscation) Rate of true positives vs. false positives for classifying memory request traces of medical disease queries. Each query is for the average cost of different admitting patient diagnoses. This shows our modeled attacker's classification has near perfect accuracy when observing the original queries execution.

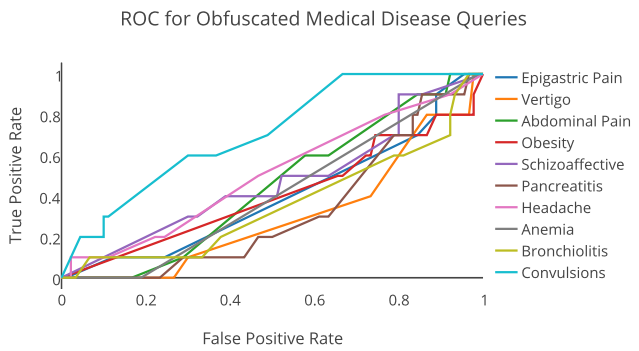

Figure 6: (Post-obfuscation) Rate of true positives vs. false positives for classifying obfuscated memory request traces of the same medical disease queries using a privacy parameter of $\epsilon = 0.1$. This shows our modeled attacker's classification is much more difficult after applying our obfuscation scheme.

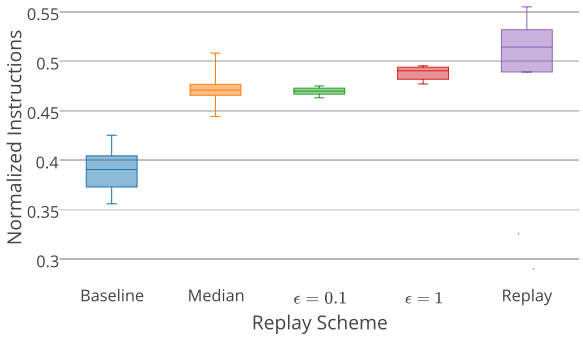

Figure 7: Instructions per cycle (IPC) slowdown across 50 different admitting patient diagnosis queries for 1 billion cycles for the four noise-addition schemes. Baseline secure (time partitioned) execution is slowest, privacy parameter $\epsilon = 1$ is better than $\epsilon = 0.1$, which is almost as slow as the perfectly private median execution, and reasonably close to simply replaying the original execution (i.e. non-secure but close to (statistical) upper limit for STOPP-Q.)

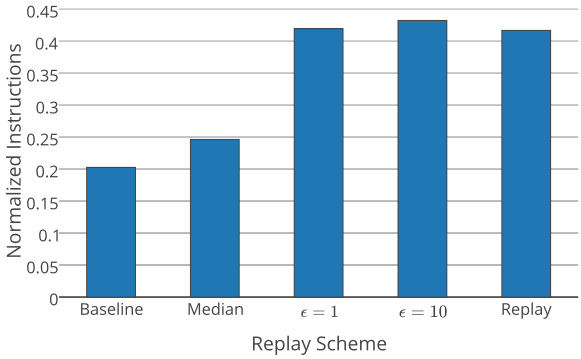

Figure 8: IPC slowdown for model runs for 10 Moses translations of 100 sentences from Don Quixote from English to Spanish (not shown as box plot due to low variance). Baseline time-partition and median replay are the slowest, while the noisy replays are roughly equal to the statistical best case of replaying original trace.

private.

Figure 9 shows the results of our ZSim runs of 10 graph inputs and 8 algorithms from Ligra [66], grouped by algorithm. The overall trend is that of the replay trace having instructions per cycle (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 non-determinism.

7. HARDWARE IMPLEMENTATION

We implement our privatized queue (shown in Figure 4) in the Chisel hardware generation language [72] and incorporate it into the RISC-V Rocket Chip Generator [31, 32]. We leverage Rocket's existing accelerator command interface that sends custom instructions from the core to the accelerator and can receive responses to write results into ISA-level registers. In addition, Rocket contains interfaces to connect an accelerator to the L1 data cache, L2 cache, floating point unit, and the hardware page table walker. We use Rocket's accelerator command and L2 cache interfaces to integrate a SHA3 hashing unit as shown in Figure 10 with each core contending for usage of the accelerator. The victim and the attacker each run an application that sends hashing requests to an instantiation of our privatized

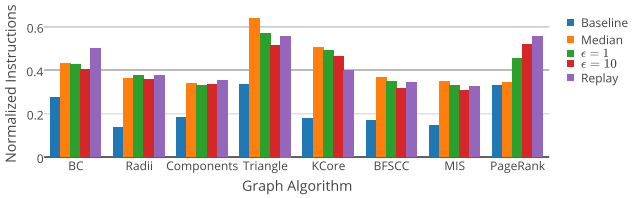

Figure 9: IPC slowdown for each of the 10 Ligra [66] graph algorithms on 10 real-world graphs [70] (not shown as box plot due to low variance). Baseline secure (time-partitioning) system is slower than STOPP-Q's executions, with PageRank showing the most obvious trend from large noise (and low performance) to less noise (and better performance). Exceptions in the trend are due to noisy or median replays' destructive interference with program's actual phases.

queue configured with 16 entries and a 128B SRAM to hold the noisy request frequencies. In Table 4 we show the extra implementation overhead of using our private queue compared with an unmodified queue on an FPGA is only a 0.4% increase in Slice LUTs and 1 additional Block RAM. In addition, there was no impact on the critical path of the Rocket Chip design. When running hashing on 10 of the input sentences from Don Quixote (as used in the Moses evaluation), we see an average overhead increase per hash of 1.57x.

	Slices	BRAMs
Rocket	46626 (87.64%)	13 (9.29%)
Rocket with Private Queue	46815 (88.00%)	14 (10.00%)

Table 4: FPGA synthesis results before and after privatizing the accelerator queue on a Xilinx Zinq-7000.

8. CONCLUSION

STOPP-Q is the first step in a new direction — replacing perfect security and low performance with a principled way to quantifiably trade off security for performance, specifically targeting the problem of obfuscating utilization traces. Programs that have similar behavior across inputs, but variations in phase behavior over time, stand to gain most from STOPP-Q's noise addition. STOPP-Q will automatically learn the common behaviors (such as loading a database or streaming in a map phase) and the differences over time (sequence of phases) and tune each execution to be significantly more efficient than normalizing its utilization trace.

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 an intelligent attacker (i.e., an extensively optimized classifier). This classifier, which could almost perfectly classify traces to inputs before noise addition, had its classification accuracy reduced to effectively ran-

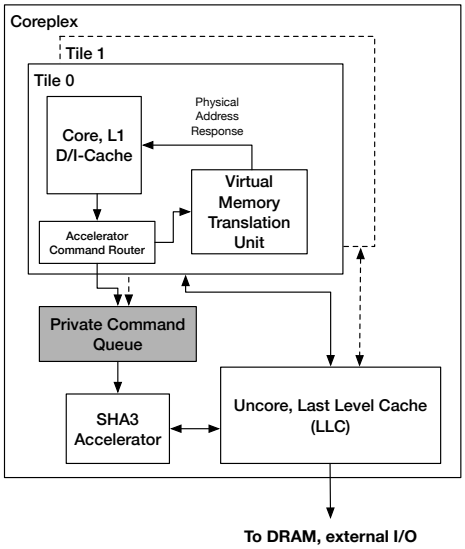

Figure 10: Another example system where the queue under attack holds requests from each core to a SHA3 accelerator. The virtual addresses for the hash input and output are translated inside the core and then sent to the command queue modified to shape the victim's usage according to our obfuscated model. The SHA3 accelerator directly accesses the last-level cache to perform the hashing operation.

dom guessing. Additionally, we show through detailed architectural simulation that shaping memory traffic outperforms a secure system that statically partitions memory bandwidth. Finally, we concretize our noise addition scheme as a *secure queue* data structure in a hardware description language and demonstrate its use to share an accelerator in a multi-core RISC-V system.

In conclusion, this work raises interesting questions about other problems in systems security where time-series based privacy-preserving algorithms might apply.

9. REFERENCES

[1] Intel Corp. Software guard extensions programming reference. `https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf`, 2014.

[2] Advanced Micro Devices Inc. Amd memory encryption. `http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf`, 2016.

[3] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In Ruby B. Lee and Weidong Shi, editors, *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013.

[4] 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.

[5] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pages 313–328. IEEE Computer Society, 2011.

[6] 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.

[7] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 719–732. USENIX Association, 2014.

[8] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. PHANTOM: practical oblivious computation in a secure processor. In Sadeghi et al. [73], pages 311–324.

[9] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 159–173. USENIX Association, 2012.

[10] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M. Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 281–292. ACM, 2012.

[11] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. Timing channel protection for a shared memory controller. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*, pages 225–236. IEEE Computer Society, 2014.

[12] Andrew Ferraiuolo, Yao Wang, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. Lattice priority scheduling: Low-overhead timing-channel protection for a shared memory controller. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016* [74], pages 382–393.

[13] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. Understanding contention-based channels and using them for defense. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 639–650, Feb 2015.

[14] Dmitry Evtyushkin and Dmitry V. Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 843–857. ACM, 2016.

[15] 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.

[16] Mehmet Kayaalp, Nael B. Abu-Ghazaleh, Dmitry V. Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*, pages 72:1–72:6, 2016.

[17] 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* [74], pages 406–418.

[18] Johan Agat. Transforming out timing leaks. In Mark N. Wegman and Thomas W. Reps, editors, *POPL 2000, Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Boston, Massachusetts, USA, January 19-21, 2000*, pages 40–53. ACM, 2000.

[19] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 431–446. USENIX Association, 2015.

[20] Ashay Rane, Calvin Lin, and Mohit Tiwari. Secure, precise, and fast floating-point operations on x86 processors. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 71–86. USENIX Association, 2016.

[21] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA*, pages 118–129. IEEE Computer Society, 2012.

[22] Wei-Ming Hu. Reducing timing channels with fuzzy time. In *IEEE Symposium on Security and Privacy*, pages 8–20, 1991.

[23] Fangfei Liu and Ruby B. Lee. Random fill cache architecture. In *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014* [75], pages 203–215.

[24] Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12, pages 3–8, New York, NY, USA, 2012. ACM.

[25] 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.

[26] Yanqi Zhou and David Wentzlaff. Mitts: Memory inter-arrival time traffic shaping. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 532–544, Piscataway, NJ, USA, 2016. IEEE Press.

[27] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. Camouflage: Memory traffic shaping to mitigate timing attacks. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017*, HPCA '17, Piscataway, NJ, USA, 2017. IEEE Press.

[28] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. Side-channel vulnerability factor: A metric for measuring information leakage. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 106–117, Washington, DC, USA, 2012. IEEE Computer Society.

[29] Tianwei Zhang, Fangfei Liu, Si Chen, and Ruby B. Lee. Side channel vulnerability metrics: The promise and the pitfalls. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 2:1–2:8, New York, NY, USA, 2013. ACM.

[30] Texas Department of State Health Services. Hospital inpatient discharge public use data file. `http://www.dshs.texas.gov/THCIC/Hospitals/Download.shtm`,

2016.

[31] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste AsanoviÄĞ. The risc-v instruction set manual, volume i: User-level isa, version 2.1. Technical Report UCB/EECS-2016-118, EECS Department, University of California, Berkeley, May 2016.

[32] Krste AsanoviÄĞ, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[33] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 533–549. USENIX Association, 2016.

[34] Andrew Baumann, Marcus Peinado, and Galen C. Hunt. Shielding applications from an untrusted cloud with haven. In Jason Flinn and Hank Levy, editors, *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, pages 267–283. USENIX Association, 2014.

[35] Henry Cook, Miquel Moretó, Sarah Bird, Khanh Dao, David A. Patterson, and Krste Asanovic. A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness. In Mendelson [76], pages 308–319.

[36] Yinqian Zhang and Michael K. Reiter. Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In Sadeghi et al. [73], pages 827–838.

[37] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore J. Stolfo. On the feasibility of online malware detection with performance counters. In Mendelson [76], pages 559–570.

[38] Meltem Ozsoy, Khaled N. Khasawneh, Caleb Donovick, Iakov Gorelik, Nael B. Abu-Ghazaleh, and Dmitry Ponomarev. Hardware-based malware detection using low-level architectural features. *IEEE Trans. Computers*, 65(11):3332–3344, 2016.

[39] Mikhail Kazdagli, Vijay Janapa Reddi, and Mohit Tiwari. Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *Proceedings of the 49th International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwain, October 15-19, 2016*, 2016.

[40] Jie Chen and Guru Venkataramani. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014* [75], pages 216–228.

[41] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. Unsupervised anomaly-based malware detection using hardware features. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*, volume 8688 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2014.

[42] Tianwei Zhang and Ruby B. Lee. Cloudmonatt: an architecture for security health monitoring and attestation of virtual machines in cloud computing. In Deborah T. Marr and David H. Albonesi, editors, *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, pages 362–374. ACM, 2015.

[43] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*, pages 361–372. IEEE Computer Society, 2014.

[44] 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.

[45] Cynthia Dwork. Differential privacy: A survey of results. In *Proc. of TAMC*, 2008.

[46] Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 19–30. ACM, 2009.

[47] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 429–438. IEEE, 2013.

[48] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Secure multi-party differential privacy. In *Advances in Neural Information Processing Systems*, 2015.

[49] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. *Journal of Machine Learning Research*, 17(17):1–51, 2016.

[50] Paul Cuff and Lanqing Yu. Differential privacy as a mutual information constraint. In *Proc. of ACM CCS*, 2016.

[51] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

[52] Quan Geng and Pramod Viswanath. The optimal mechanism in differential privacy. In *Proc. of IEEE ISIT*, 2014.

[53] 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.

[54] 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.

[55] Isca2017 submission 448 comparison to camouflage. `https://github.com/anonpapsub/QueuePrivacy/blob/master/camo_comparison.pdf`, 2016.

[56] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[57] Yun Gao, Ioannis Kontoyiannis, and Elie Bienenstock. Estimating the entropy of binary time series: Methodology, some theory and a simulation study. *Entropy*, 10(2):71–99, 2008.

[58] Yun Gao, Ioannis Kontoyiannis, and Elie Bienenstock. From the entropy to the statistical structure of spike trains. *CoRR*, abs/0710.4117, 2007.

[59] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: Privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 349–360, New York, NY, USA, 2012. ACM.

[60] Jaewoo Lee and Chris Clifton. How much is enough?

choosing ϵ for differential privacy. In Xuejia Lai, Jianying Zhou, and Hui Li, editors, *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, volume 7001 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2011.

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

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

[63] 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.

[64] Pimwadee Chaovalit, Aryya Gangopadhyay, George Karabatis, and Zhiyuan Chen. Discrete wavelet transform-based time series analysis and mining. *ACM Comput. Surv.*, 43(2):6:1–6:37, February 2011.

[65] 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.

[66] 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.

[67] 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.

[68] Daniel Sanchez and Christos Kozyrakis. Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In Mendelson [76], pages 475–486.

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

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

[71] 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.

[72] Jonathan Bachrach, Huy Vo, Brian C. Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanovic. Chisel: constructing hardware in a scala embedded language. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, pages 1216–1225. ACM, 2012.

[73] 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.

[74] *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016*. IEEE Computer Society, 2016.

[75] *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014*. IEEE, 2014.

[76] Avi Mendelson, editor. *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*. ACM, 2013.