Side-Channel Vulnerability Metrics: SVF vs. CSV

John Demme and Simha Sethumadhavan
Computer Architecture and Security Technologies Lab
Department of Computer Science, Columbia University
{jdd,simha}@cs.columbia.edu

ABSTRACT

Recently two papers have been published on empirically measuring side-channel leakage in processors. The first paper introduced a framework for measuring side-channel leakage called "Side-Channel Vulnerability Factor" (SVF). SVF used phase correlation between victim and attacker programs to quantify leakage. A subsequent paper opposed some of the claims made in the SVF paper and introduced another metric, "Cache Side-channel Vulnerability" (CSV). CSV uses the same concept of measuring correlation between victim and attacker, but instead proposes using direct correlation in place of phase correlation. Another major difference between SVF and CSV is the scope of leakage measurement – CSV is defined to apply to only cache leakage, whereas SVF can be applied to multiple components within a processor. The CSV authors argue that applying SVF yields conclusions which contradicts what they term to be ground truth. In addition to these differences, the two papers used different experimental setups and thus their results were not directly comparable.

This paper deconstructs the differences between SVF and CSV. We first provide a general overview of side-channels and background on their modeling and measurement. We then examine the differences between SVF and CSV both quantitatively and qualitatively using a common experimental setup. Finally, building on our examination of differences, we review and rebut claims made in the CSV paper against the SVF framework and metrics.

1 Introduction

Side-channel vulnerabilities allow nefarious users to circumvent traditional permissions, obtaining access to private information. They could and do exist in a variety of different systems. Processor caches shared between different processes have been demonstrated to leak information about encryption keys [16, 17, 9]. Spikes in power usage on mobile processors can also leak information about encryption operations [15, 12]. Even shared networks have been used to obtain information about other users' encrypted communications [5].

Perhaps the most pernicious issue regarding side-channels is that while it may be possible for many system components to leak information, we do not know which ones or how much they leak. There may exist side-channels being leveraged in the wild of which we are unaware. It is, therefore, difficult to understate the importance of methods to find, quantify, and evaluate improvements to these side-channels.

At ISCA 2012, a paper entitled "Side-Channel Vulnerability Factor: A Metric for Measuring Information Leakage" presented a framework for measuring the potential leakage through systems [6].[1] The SVF paper proposed to estimate leakage based on the correlation of phases in victim execution to phases which an attacker can observe. Further, the authors state that leakage is a system-level property and thus should be measured in that context, devising their framework for that purpose. An underlying theme in the paper is an argument against single-component (standalone) measurement of leakage.

More recently, at HASP 2013 (a workshop co-located with ISCA), a paper entitled "Side Channel Vulnerability Metrics: the Promise and Pitfalls" presented a response to the original SVF paper, wherein the authors refuted many of the SVF paper's claims and introduced a simpler, alternate metric, Cache Side-channel Vulnerability (CSV) [26]. CSV eliminates the phase detection techniques proposed by SVF and instead directly computes Pearson correlation between victim and attacker execution data. By design, CSV restricts itself to measuring only leakage through caches. The authors conclude that CSV metric results are consistent with what they term to be ground-truth expectations of leakage.

The SVF and CSV papers agree on some issues and disagree on others. Both papers adopt an empirical approach to side-channel characterization (as opposed to mathematical modeling). In particular, both use a metric that compares victim execution to attacker observations. There are, however, at least two fundamental differences between SVF and CSV. First, the metric for comparison: SVF uses phase correlation whereas CSV uses direct correlation. Second, CSV measures leakage through the cache only whereas SVF can measure leakage through multiple components like both the cache and pipeline simultaneously. Additionally, in demonstrating their ideas, the authors of the respective paper used different experimental setups: they differ in victim and attacker application, their inputs, and simulator.

The HASP'13 paper raises some interesting questions. First, CSV is derived from an intuitive perspective on cache side-channel leakage; accordingly, their results are easy to understand. Is the simplified CSV perspective sufficient to understand cache side-channels? Second, CSV differs from SVF in several significant ways, leading to vastly different results. Which of these differences are responsible for which differences in their results? We examine these questions quantitatively by normalizing the experimental infrastructure to match as closely as possible and present experiments which resolve some of these lingering questions. In addition to these questions, the HASP'13 paper contains some inaccuracies about the original SVF paper which we clarify and correct in this paper.

We begin this paper with an introduction to side-channels (Section 2) then a background on modeling and measuring side-channel leakage (Section 3). Next, we quantitatively and qualitatively examine the differences between CSV and SVF with a deconstruction of their component properties and a set of experiments measuring

[1] Full disclosure: this paper's authors were also two of the authors of the SVF paper.

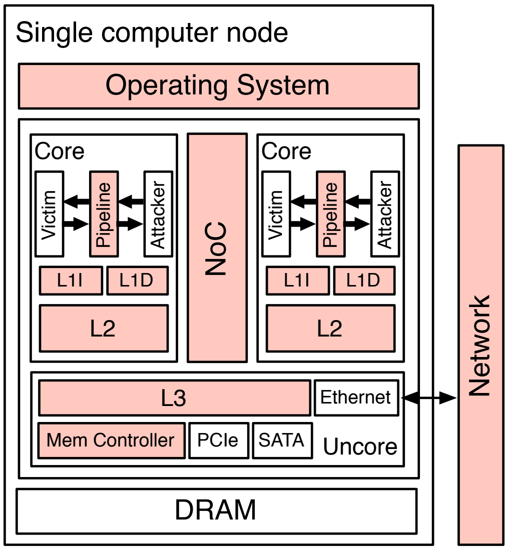

Figure 1: When an attack and victim execute on the same processor, a wide variety of components may leak sensitive information. The components highlighted are known to be exploitable with demonstrated attacks. Could the other components leak?

SVF, CSV, and a nearly exhaustive set of derivative metrics which incorporate properties from both (Section 4). Building on this analysis, Section 5 reviews the claims about SVF which were discussed the CSV paper. Finally, we conclude with a discussion of open problems in side-channel metrics and conclusions from this study in Sections 6 and 7 respectively.

2 Side-Channels: A General Introduction

Most actions have side-effects which can be used to infer private information. For instance, the creaks as one walks in an old apartment alerts neighbors to one's presence. The same side-effects occur in computing and in fact attackers can use these "side-channels" to obtain sensitive information. For example, the behavior of encryption software is partially a function of the encryption key it is using; thus, an attacker may be able to recover the encryption key by measuring how long it takes for the encryption to complete or how much power each encryption round consumes.

Side-channel leaks occur at all levels in the computing stack. The time between memory allocations, or the amount of memory allocated may leak information about a web page being rendered [11]. The size and frequency of the network packets may be used to break encrypted VoIP conversation [25]. Concurrently running encryption alongside another program on the cloud may leak information about the keys [28]. It is expected that applications, operating systems, virtual machine monitors, and processors leak information creating potential side-channels. In the way of introduction, in this section we first describe a side-channel attack in generic terms then describe a concrete attack.

2.1 Generic Processor Side-Channel Attack

To describe a generic processor side-channel attack we need at least two entities: a *victim* program, and an *attacker* program.

Victim: The victim program is the program from which the attacker is trying to steal information. In most of the side-channel attacks, the victim program tends to be a cipher such as AES or RSA. In fact the practice of attacking ciphers is so common that the name side-channel attacks has become synonymous with attacks on cryptographic programs. Technically speaking, however, the victim program can be any program. For instance, in some recent attacks a web browser rendering pages and an arbitrary process using address-space layout randomization have been victimized.

Attacker: The attacker program is the program that is trying to extract sensitive information from the victim. This program is often carefully tailored to exploit precisely identified leakages in one or more processor structures. It is convention to name the attack type based on the structure that the attacker program is targeting. For instance, if the attacker program is targeting the L1-D cache, the side-channel attack may be referred to as a cache side-channel attack.

However, be warned that the convention of naming attacks after the structure is somewhat superficial. Although the attacker may be trying to extract leakage from a particular structure (*e.g.,* cache side-channel), she may be indirectly aided by leakage from a different processor structure (*e.g.,* pipeline). Thus, it is important to differentiate the attacker program, the actual source(s) of leakage, and the attacker's perception of the source of leakage. We will revisit this distinction with a concrete example in the next subsection.

In addition to these two central entities, there are two additional factors that have to be included in any generic description of a processor side-channel attack.

Synchronization Strategy: To extract leakage from the victim, the attacker can scavenge the leaky structure simultaneously with the victim or just after a victim has used a structure. Attackers that operate concurrently (in unison) are referred to as asynchronous attackers, while attackers that take turns with the victim are referred to as synchronous attackers. Please note that the synchronous and asynchronous adjectives are exactly the opposite of their typical English usage.

Denoising Strategy: Any data extracted by the attacker during the attack is likely to be noisy due to real system effects such as interrupts, inherent uncertainty from scheduling, autonomous DVFS etc., and also due to nature of leakage being exploited (for instance, the cache may not be accessible at precisely at the time when it is used by a victim). To recover the sensitive information from the raw data the attacker has to perform some denoising. Often the ability to extract signal from noisy measurements determines the efficacy of the attack.

Last but not the least, we need leaky processor structures to carry out side-channel attacks. Figure 1 shows a typical microprocessor microarchitecture and highlights the components are considered exploitable. In the last decade several papers have published attacks that utilize information available through other shared microarchitectural structures including I- and D-caches [17, 1], branch predictors [2], SMT functional units [3], TLBs [10], network-on-chips [20], and second- and last-level caches [27].

Our prediction is that despite a lack of published attacks, many other on-chip shared structures including dependence predictors, various intermediate buffers, energy management units, prefetchers, page cache structures, a variety of on-chip controllers, transactional memory primitives, and pretty much any microarchitectural structure that is shared can leak information, and that this leakage can be used to construct or aid in side-channel attacks.

2.2 Demonstrative Example: A "Cache" Attack

In this section, we detail the operation of a particular type of side-channel attack – an asynchronous prime-and-probe cache side-channel attack – on the RSA encryption algorithm [17]. This particular attack is useful as a demonstrative example to develop intuition on the operation of side-channels.

The attack against RSA exists for two reasons: First, implementations of RSA exhibit microarchitectural behaviors which differ depending on the encryption key. In the case of RSA, OpenSSL contains branches which are data-dependent on the key. Second, an attacker can observe – either directly or indirectly – these microarchitectural behaviors. An asynchronous prime-and-probe attacker observes microarchitectural behavior by continuously scanning a shared cache with memory instructions, detecting when they miss in the cache and inferring that the victim evicted their cache line in that case.

We will first detail the operation of RSA which enables the leak followed by an explanation of the attacker which can exploit the leak.

The Victim RSA operations are defined rather simply as modular exponentiation:

$$c \equiv m^e \pmod{n} \tag{1}$$

Wherein m is the plaintext, (n, e) is the key, and c is the ciphertext. This operation, however, is deceptively simple because m, e, and n can be relatively large numbers and exponentiation is a complex operation, preventing RSA from being computed via a single CPU operation. Instead, RSA implementations often use exponentiation by squaring, defined recursively as:

$$x^e = \begin{cases} x(x^2)^{\frac{e-1}{2}} & \text{if } e \text{ is odd} \\ (x^2)^{\frac{e}{2}} & \text{if } e \text{ is even} \end{cases} \tag{2}$$

This algorithm is interesting and leaky in large part because it uses cases based on e. As a result, the code contains a branch which is data-dependent on the encryption key. Futher, the 'then' and 'else' bodies following this branch contain non-trivial and different operations involving function calls to a math library. As a result, the instructions being executed and memory addresses being executed by the victim are significantly altered by the encryption key. The instructions being issued affect components like the I-Cache and the pipeline functional units. The memory addresses being accesses will affect the victim's cache fingerprint. If either of these microarchitectural effects can be sensed, a leak is possible.

The Attacker, in Theory Based on the intuition that an RSA operation's cache fingerprint changes over time in a pattern depending on the encryption key, an attacker may attempt to get an approximation of this fingerprint and use it to obtain an approximation of the encryption key. If the attacker and victim share a cache, it seems intuitive that an attacker can sense this fingerprint using contention for cache lines.

The synchronous (rather than asynchronous) prime-and-probe attack is easier to understand, so we will first explain its operation and then convert it into an asynchronous attack. First, the attacker "primes" the cache by issuing loads to all of the cache lines, evicting all of the victim's data. Next, the attacker allows the victim to execute for some time, during which the victim will issue stores and loads to the cache, evicting some of the attacker's cache lines. Finally, the attacker scans the cache again but times the loads to each cache set. Loads which take longer indicate that the attacker experienced one or more misses in that cache set, which further implies that the victim issued a memory access to that set. The attacker then lets the victim run for some period and repeats the scan, ad infinitum.

After recording cache set load latencies in each scan and executing many scans over the entire execution of the RSA operation, the attacker attempts a complex post-processing step to decode her load latencies to hits and misses, then patterns of hit and misses to encryption key bits. The complexity of this step often depends on quality of observations – noisy data are more difficult to decode.

The synchronous attack model assumes that the attacker can arbitrarily interpose itself into the victim's execution, interrupting it at will. Sometimes this is feasible [9], other times it is not. An attacker can also execute an asynchronous attack wherein he runs in parallel with the victim and shares a cache and possibly a pipeline via simultaneous multi-threading. The asynchronous attack works the same way as the synchronous attack, but may reduce the wait time in between scans to zero because the victim was given a chance to run while the attacker was issuing loads. Despite possible noise issues related to executing simultaneously with the victim, the asynchronous attack has been demonstrated [17].

The Attacker, in Practice Extracting leaked data from cache line ownership is ostensibly how attackers in practice operate. There is, however, one assumption baked into the this description: that their measurements – noisy or not – are affected only or primarily by cache hits and misses. Microarchitectures are more complex than that, however. In particular, for the asynchronous attacker sharing a pipeline with the victim, a host of interference/contention with the victim can affect the attacker's observations. Contention for functional units, load/miss buffer slots, cache interconnect bandwidth, and memory bandwidth all affect cache load latency, just to name a few. From the perspective of the attacker, this differentiation is irrelevant – they either obtain data which can be exploited or not – which components affect these data is academic to them. In terms of modeling, measuring, and reducing processor side-channel leakage, however, knowing which components positive or negatively affect leakage is critical. If it turns out the attacker's measurements are mostly determined by contention for pipeline functional units (for example) then statically partitioning the cache (thus isolating the attacker from being evicted by the victim) will not affect leakage and could in fact increase leakage. Given this insight, it is possible that "cache side-channel" may be a misnomer.

3 Background: Discovering Side-channel Leaks

While there are many solutions [23, 21, 8, 24, 19, 14] to mitigate or block side-channel leaks until very recently there was no processor design time technique to test a microarchitectural idea for its vulnerability to side-channel information extraction. Design-time techniques for identifying leaks are extremely valuable, because we can then use these techniques to identify and fix leaks before processors hit the market.

3.1 Proving Non-existence of Side-Channels

One approach to testing a microarchitectural idea for leakage is to use Gate-level Information Flow Tracking techniques to prove a security property known as non-interference [24]. Non-interference is a strict binary property. The main idea is to use a variant of taint tracking and white-box validation to prove that two information flows never interfere with each other. The technique itself is easy to apply and scalable, but strictly non-interfering systems impose high-cost and overheads that limit the widespread application of these systems. Further, the analysis itself needs an RTL description of the design. It would, however, be desirable to identify leaks much earlier in the design cycle.

3.2 Accepting Side-channels Will Exist But Estimating Risk

While non-interference was the golden standard for security in early days of security, it is accepted today that side-channel security should be considerd a trade off between leakiness and associated risks. This shift in thinking underscores the idea that security is a continuous property rather than a binary one. Pending major breakthroughs in low-overhead, low-cost, non-interfering systems (which has eluded us for 20+ years) the leak-risk tradeoffs mindset offers a pragmatic solution.

To estimate the leakiness of systems, there have been some efforts to model and measure side-channel leakage. Köpt and Basin [13] then Domnitser et al. [7] introduced methods of modeling and bounding side-channel leakage. More recently, Demme et al. [6] followed by Zhang et al. [26] proposed experimental metrics for measuring leakage. All of these different techniques for analyzing side-channels have pros and cons which we will review next.

Mathematical Modeling to Estimate Leaks In modeling based approaches, the system being analyzed is itself mathematically modeled in a relatively simple form. To continue the cache example, Domnitser et al. [7] model ownership of cache lines (and therefore contention for the lines), several basic properties of caches like set associativity, and some of the characteristics of both the victim and attacker executions like the amount of time in between repeat cache line accesses.

Using relatively simple mathematical models enables modeling-based analyses to apply sound reasoning to the leakage problem. For instance, it may be possible to prove non-interference (and thus zero leakage) between victim and attacker – this would be very easy to show for a statically partitioned cache and may be possible for certain cache replacement policies. In Domnitser et al. [7], a probabilistic model is defined which allows computation of the probability that attackers can view critical cache accesses, an interesting, intuitive metric.

The major drawback to modeling approaches for analysis is that they require relatively simple mathematical models of the system being analyzed to enable said analysis – if the models are too complex, mathematically proving the (non-)existence of a side-channel becomes difficult and perhaps intractable. In reality, however, these systems are tremendously complicated and defy simple modeling. Domnitser et al. [7], for example, do not model complex replacement policies, prefetching, load-miss buffers, or any other cache feature beyond the cache lines themselves. Adding these features to their model is likely infeasible – they would complicate it beyond the point where one could reason about it, thus limiting conclusions.

Empirically Estimating Leakage Using Simulators To avoid having to accurately model complex systems, measurement approaches treat systems as black boxes, as we see in Figure 2. Instead of being able to analyze a system's model, they operate experimentally, monitoring the execution of a system to calculate a metric related to leakage [26, 6]. This is the same methodology that most of computer architecture uses – we measure power, energy, performance, and a host of microarchitectural metrics to judge system performance. As a result, measurement methods work on systems which are arbitrarily complex and thus their accuracies have no limit (in theory).

As an experimental approach, leakage measurements depend on the same combination of factors as other measurements such as all of the system components, the code being executed, inputs to the program, and the application an attacker is executing. Unlike other microarchitectural measurements, however, side-channel leakage suffers from one additional problem: using an average over many programs and runs may not be sufficient. While averages are relevant for performance, in the adversarial environment of security attackers can often leverage worst-case scenarios. Is the same true for leakage? What sort of worst-case scenarios can be forced by attackers? Answers to these questions are unclear. Even if they were known, it may not be possible for experimental methods to analyze worst-case situations. While measurement methods have the capability to account for all aspects of complex systems – in direct contrast to modeling – their inability to measure all possible scenarios could mean that they miss important leaks.

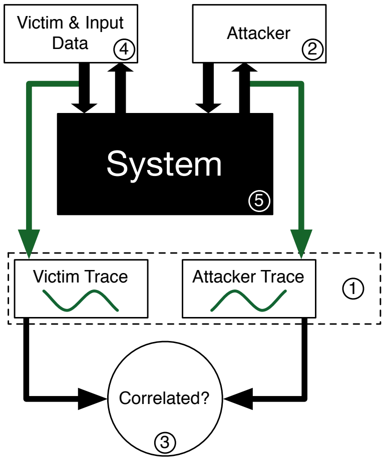

Figure 2: Proposed leakage measurement techniques model the system as a black box and collect traces from the victim and attacker's execution. Then, the determine whether or not the two traces are correlated. As a result, measurements are a function of the victim program, victim's inputs, the attacker, and the system.

4 SVF vs. CSV: A Comparison

To estimate leakage, two recent papers (SVF and CSV) use the experimental methodology shown in Figure 2. Specifically, both use a pair of time-series "traces" – one representing private data used in a victim's execution, the other representing data being gathered by an attacker – to model a particular run in the system being evaluated. For example, the victim trace could represent bits from an RSA encryption key being used by OpenSSL and the attacker trace could contain load completion latencies from its scan through the cache. The challenge then is to determine how much of the private information in the victim's trace could be recovered from the attacker's trace. Informally, the challenge is determining whether or not the two traces are correlated – if the attacker's observations are affected by the victim's trace, then those observations will have some correlation to the victim's trace. Correlation, however, can be measured in a variety of different ways depending on the context.

SVF proposes that traces first undergo a form of pattern recognition in the form of phase detection. In particular, SVF applies the phase detection from SimPoint phase analysis [18] and constructs self-similarity matrices for each trace. SVF then calculates the Pearson correlation coefficient between the two self-similarity matrices and uses this correlation as a metric for leakage. Intuitively, this approach reveals how well phases in the victim application are reflected in observations an attacker has made. This use of phase correlation is interesting for several reasons. First, it enables one to compute correlation between two traces with different types (*e.g.,* memory addresses vs. cache misses). Second, it emulates the learning of patterns in observations which occur as a result of repeated events in the victim.

More recently, Zhang et al. [26] propose a new metric: Cache Side-channel Vulnerability (CSV). CSV adopts the same methodology as SVF in using experimentally derived traces, but specifically defines two particular traces which are different from those

		Multi-Component SVF	**Cache-Only SVF**	**CSV (Our impl.)**	**CSV ([26])**
Data collected for & presented in		This Paper	This Paper	This Paper	CSV Paper [26]
Leakage potentially occurs through		All simulated components	Cache	Cache	Cache
① Scope	Victim Trace	Memory Accesses	Cache Set Accessed	Cache Set Accessed	Cache Set Accessed
	Attacker Trace	Cache Set Load Latencies	Cache Set Missed	Cache Set Missed	Cache Set Missed
② Attacker	Asynchronous	✓	✓	✓	✗
	Synchronous	✓	✓	✓	✓
	Sync. Interval	5000	5000	5000	100
③	Correlation Method	Phase Correlation	Phase Correlation	Direct Correlation	Direct Correlation
④	Victim Application	OpenSSL RSA	OpenSSL RSA	OpenSSL RSA	AES
⑤	Simulator	Home grown	Home grown	Home grown	GEM5 [4]

Table 1: Characteristics of vulnerability metrics

used in the SVF paper's case study. In particular, CSV defines the victim trace to be whether or not the victim accessed each cache set during a particular epoch and the attacker trace to be whether or not the attacker experienced a cache miss in each cache set during each epoch. As a result of using traces which are similar in data type (specifically cache related events), CSV avoids using phase detection techniques. Instead, they can directly compute Pearson correlation between the traces, which they use as a leakage metric.

Although both are intended to measure side-channel leakage, SVF and CSV are very different metrics and reflect different perspectives on side-channel leakage. Accordingly, the results obtained by the two papers are significantly different. Next, we analyze the differences between SVF and CSV in both qualitatively and quantitatively.

SVF, CSV, and the experiments their authors have published differ in five respects, as summarized in Table 1:

① **Scope** The choice of traces directly impacts the scope of leakage measurement. SVF defines traces which measure the correlation between the memory addresses being accessed by a victim and load latencies to each cache set which an attacker can observe. This captures potential leakage in any component which affects execution of memory instructions. In contrast, CSV defines victim and attacker traces which contain whether or not cache sets[2] were accessed by the victim and whether or not cache sets experienced misses by the attacker. By construction, CSV measures only leakage through cache line ownership.

② **Attacker** The SVF authors used an asynchronous attacker wherein the attacker and victim run in parallel, often sharing a pipeline via Simultaneous Multi-Threading (SMT). The CSV paper used a synchronous attacker, assuming that it was capable of interrupting the victim every 100 cycles and scanning the cache contents.

③ **Correlation Method** SVF uses a phase detection based correlation metrics which we will call "phase correlation" whereas CSV directly computes Pearson correlation between traces.

④ **Victim** The SVF paper examined OpenSSL's RSA signing operation whereas the CSV authors used an AES algorithm.

⑤ **Simulator** The SVF paper used a simulator its authors developed from scratch whereas the CSV paper used a modified version of GEM5 [4].

Cache Size	L1=32K, L2=256K, L3=8MB
Line Size	64B or 8B
Associativity	8-way or 4-way
SMT	On – victim & attacker share a core
Prefetcher	None
Attacker Scan Style	Scans all sets in order
Partitioning	None, Dynamic, Static
Random Eviction	None, 5%, 10%, 50%
Hashing Scheme	Low Bits, XOR, PRS-100, RPCache

Table 2: Cache configurations for the results shown in Figure 3.

4.1 Experimental Methodology

To quantitatively tease out the differences between SVF and CSV, we duplicate the experiments from the CSV paper. We also create an SVF metric which measures leakage only in the cache by using the same traces as CSV. Finally, the attacker style (synchronous vs. asynchronous) is also varied. These combinations are used to determine precisely which properties of the metric and experimental setup affect results and how. All were implemented in the newest version of the custom simulator used in Demme et al.'s SVF paper [6] (the details of which can be found in that paper). To limit the number of combinations, we ran with the OpenSSL RSA victim which was used in the SVF paper. Table 1 shows these metrics and compares them to the CSV metric and evaluation environment used in Zhang et al.'s work [26].

Metrics To determine what effect differences in metric have on results, we present three different metics: (1) the original SVF metric defined by Demme et al. using phase correlation and traces defining a multi-component scope, meaning they account for leakage in many different components. (2) Cache-Only SVF which uses SVF phase correlation, but defines traces in a manner identical to CSV. (3) The original CSV metric which uses direct correlation and traces which define a cache-only scope. We note that the fourth possible combination – CSV-like direct correlation with multi-component system scope – is not possible since CSV's direct correlation is incapable of handling traces of different types.

Cache Features Our baseline and variants are shown in Table 2. Most of the configuration parameters are well known options. Those options deserving further specification are explained here:

Partitioning Intuitively, splitting the cache between the attacker and victim should eliminate or reduce the side-channels since the attacker can no longer experience misses as a result of victim actions. Static partitioning allocates half of the lines in each set to each thread (victim/attacker). Dynamic parti-

[2] In this description, cache set does not mean physical cache set. Rather, it is the cache set before hashing schemes are applied.

Access Thread	Result	Eviction Line Owner	Action
Victim	Hit	-	Regular hit processing
Victim	Miss	Victim	Regular miss processing
Victim	Miss	Attacker	Swap accessed set with a randomly selected set; Invalidate victim's lines in swapped sets
Attacker	Hit	-	Regular hit processing
Attacker	Miss	Attacker	Regular miss processing
Attacker	Miss	Victim	Swap accessed set with a randomly selected set; Invalidate victim's lines in swapped sets

Table 3: RPCache PRS permutation policies.

tioning regularly determines which cache lines should be exclusively allocated to a thread and which should be shared, the same as used in the SVF paper [6].

Random Eviction Earlier work [6] implemented a feature which randomly evicts cache lines. It serves to add noise to attacker measurements, thus one would expect it to reduce SVF and/or CSV. An eviction occurs each cycle with configurable probability and uniform randomly selects a cache line to evict.

Hashing Scheme In addition to the relatively standard low bits and XOR hashing schemes, we also examine "PRS-100" and "RPCache". These two schemes use a mechanism called permutation register sets (PRS) [22] which maps virtual cache sets to physical ones, essentially allowing for changes in the hash over time. In PRS-100, we use the same PRS policy as used in Demme et al. [6] and randomly swap two sets' mappings once every 100 loads. Further, in PRS-100 the PRS mappings are applied only to the victim application as discussed in the original PRS paper [22]. RPCache [23] uses a different policy wherein PRS is applied to both threads (with different mappings for each one) and a permutation occurs at certain cache misses, as we detail in Table 3.

4.2 Comparison

With this variety of metrics and different attackers, we can quantitatively determine which factor impacts differences in results. In this subsection we examine each parameter and review its effect on leakage measurement: How does scope affect leakage measurement? Are asynchronous or synchronous attackers more powerful? Does correlation method matter? Do the details of application and simulated hardware change measurements?

The results of our simulations can be found in Figure 3. Since it explores a multi-dimensional space, the reader is requested to refer to the appropriate subset of charts mentioned in each analysis below. The first row contains results using the same measurement methods as Demme et al. [6], the bottom row contains results using CSV as defined by Zhang et al. [26] but with the same experimental setup as the other configurations. The rows in between contain data with different combinations of options for the measurement method, bridging the gap between the two extremes.

How does scope impact leakage measurement? (Charts A and B) These data indicate that the scope of measurement (*i.e.,* whether multiple component or only cache leakage is being measured) has profound impact upon the measurement. We see that multi-component SVF with an asynchronous attacker (row A) is nearly unaffected by any cache design parameter – as if the attacker were not even using the cache. When SVF's scope is narrowed to cache only (row B), however, many of the features we would expect to degrade the leakage work as intuitively expected. Statically partitioning the cache, for example, drops cache-only SVF to 0. Since both of these rows utilize phase correlation and differ only in their scope, the difference in leakage for statically partitioned caches shows that attackers *can* utilize leakage in other components even by only issuing loads.

Does attacker style impact leakage? (Charts A, B and D) If we compare multi-component SVF with an asynchronous attacker (row A) to the synchronous attacker (row D), we observe similar trends as when the scope is narrowed to cache-only (row B). We suspect the reason for this is that side-channel leakage data contained in cache line ownership is long-lived enough to be probed by a synchronous attacker's interrupts, whereas other potentially vulnerable components' states are far more transient and thus must be probed by an attacker running concurrently. Comparing the cache-scope metrics (cache-only SVF and CSV) confirms the data from multi-component SVF – in the restricted scope, trends in design features' effects tend to be similar for asynchronous and synchronous attackers.

Does correlation method impact leakage measurement? (Charts E and F) As seen of rows E and F, the trends exhibited by Cache-Only SVF and CSV are quite similar; however, one striking exception is XOR indexing wherein CSV rates its vulnerability near zero, though we would not expect XOR to offer substantial protection. CSV reports a near-zero vulnerability for XOR because a given address in the victim and attacker will be assigned to different cache sets. As a result, when the victim displaces an attacker's cache line in some set S, the attacker sees a miss in S′, since we assume that the attacker does not know the relationship between S and S′. CSV, however, measures the correlation between accesses in S and misses in S, *not* S′. If, however, the attacker is able to learn the relationship between S and S′, a very simple post- processing step will reveal significant leakage. Since SVF uses phase correlation instead of direct correlation, it – in essence – learns the mapping between S and S′.

In addition to differences in XOR vulnerability rating, we also observe that SVF finds some leakage in PRS-100 and RPCache, whereas CSV does not. This is likely because there is another form of leakage for which SVF can account but CSV cannot: aggregate numbers of misses which vary over time. Phase correlation captures this pattern, however CSV will not if the misses are continuously shifted from cache set to cache set, as they are in PRS-based schemes.

Victim, Simulator, & Attacker Interval (Row F) The victim application, simulator, and synchronous attack interval differ between Zhang et al. [26] and this paper. Any differences between the bottom row of Figure 3 and their paper's results (not reproduced here) can be attributed to these differences. Many of the trends are similar, including the fact that static partitioning eliminates leakage in all cache-only contexts. There are, however, several differences between our CSV results and theirs: (1) Dynamic partitioning lowers CSV in Zhang et al., but not here. We suspect this is a difference in dynamic partitioning algorithms. (2) Very small random eviction rates lower CSV much more quickly in our experiments. This is likely because our synchronous attacker's larger interrupt period means more side-channel noise is added in between scans. (3) XOR is rated lower by us. This is likely a result of how Zhang et al. define "cache set" in their metric; we assume they use physical cache sets for low bits and XOR but the virtual (pre-hash) sets for PRS-based schemes, in essence assuming that an attacker cannot reverse

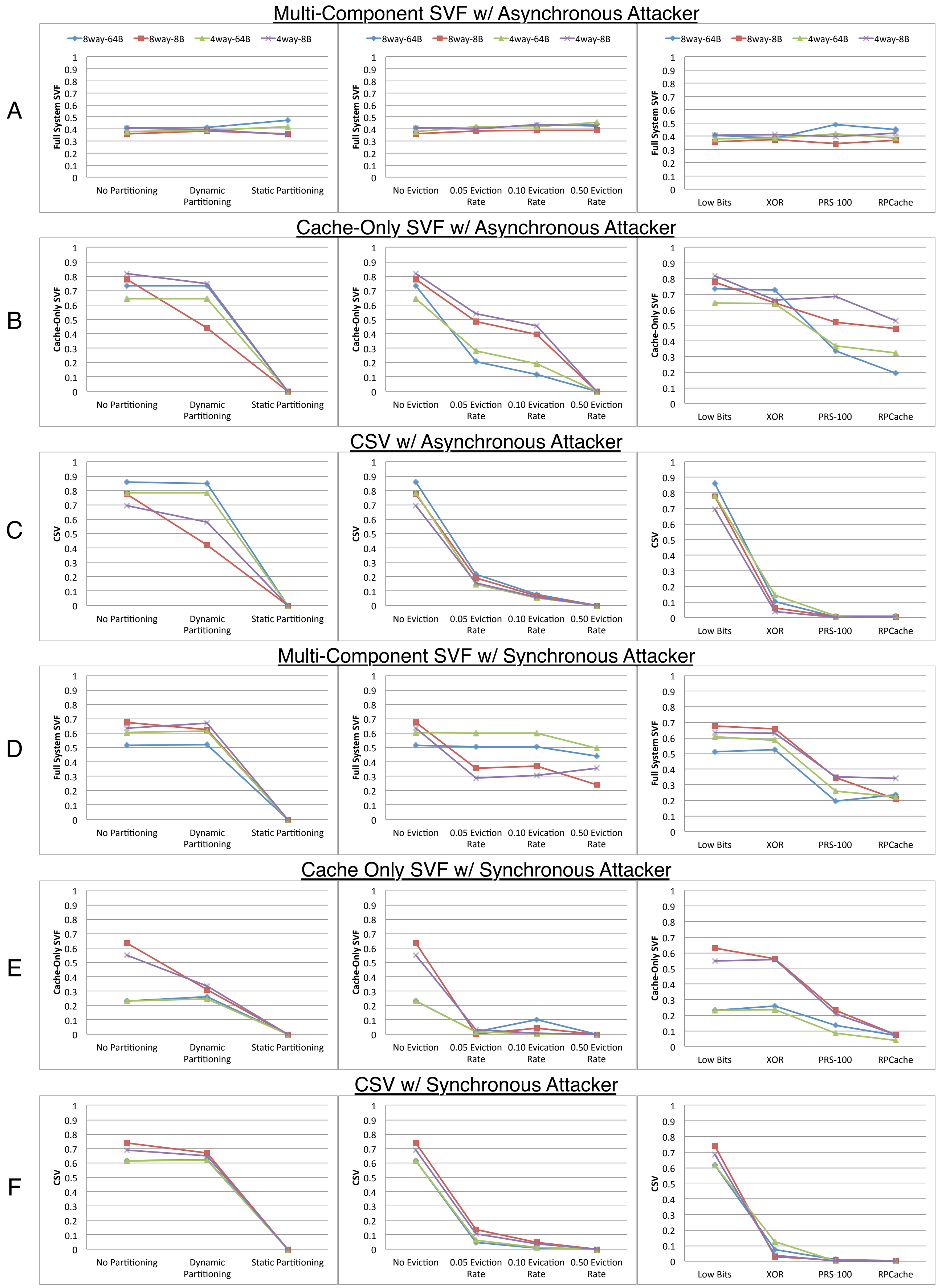

Figure 3: The effects of cache configuration and attacker types on various vulnerability metrics

PRS mapping but can reverse the XOR hash.

4.3 Conclusions

The experiments and analysis in this section provide evidence for two important conclusions: (1) While cache-only scope could be useful for debugging leakage in a cache, multi-component scope is necessary to determine broader vulnerability. Using only a cache-only scope is akin to measuring cache miss rates and assuming them to be the only determinant of processor performance. (2) CSV's direct correlation, while intuitive, assumes very little ability of attackers to post process data and can thus underestimate leakage. As a result, even schemes which provide only superficial protection (like the XOR example) can be rated as having little vulnerability. Overall, phase correlation appears to be superior in finding a superset of leaks which direct correlation discovers and is flexible enough to be applied to a variety of different scopes.

5 Discussion of Claims in the CSV Paper

In addition to proposing CSV, which we reviewed in Section 4, Zhang et al. [26] discuss some "limitations" to SVF and describe characteristics for "a good system vulnerability metric". In this section we review both.

5.1 Discussions on SVF

The CSV paper has a section entitled "Discussions on SVF" which comments on several features of SVF. Several of the limitations they discuss relate to the SVF paper's selection of traces, attacker style, and attacker capability. The previous section reviewed these topics quantitatively. Additionally, Zhang et al. [26] list several other limitations regarding SVF, which we address here.

SVF as a Framework to Define Metrics Zhang et al. [26] state in Section 3,1, Para. 1:

> "The aim of SVF is to use a single metric to reveal the information leakage of the entire system for all side-channel attacks. [...] According to Demme's SVF definition, the SVF is the correlation between the similarity matrices of two traces; let's call them X and A. Thus this SVF can ONLY evaluate the information leakage of X through A, instead of the entire system."

SVF has the ability to observe leakage through all the system components which directly or indirectly affect events at trace points X and A. Since microprocessor components are interconnected (*e.g.,* cache and pipeline), any component directly or indirectly connected to X or A could affect X and/or A. In Section 4 of this paper, we provide evidence that multiple components contribute to leakage even though the attacker may ostensibly target only one component, reinforcing the importance of this measurement feature.

The CSV paper also says in Section 3.1, Para. 2 that:

> "[...] *SVF can not be what we term a universal metric,* i.e., a single metric for all side channel vulnerabilities in a system, as claimed in [9]."

We agree that SVF is not a universal metric, and it was never claimed to be in the original SVF paper contrary to the above attribution[3]. Although not explicitly stated by Demme et al., SVF was designed as a framework with which many different metrics can be defined to measure different types of leakage throughout a system. For example, in Section 4 of this paper, we defined a new cache-only SVF metric to focus on leakage in the cache.

Phase Correlation The CSV paper takes exception to the use of similarity matrices (which are used to compute phase correlation) instead of a simpler scheme like direct correlation. The argument against it is presented in Section 3.2, Para. 1 as:

> "[...] it may not be appropriate to calculate the correlation of two similarity matrices to compare their traces. [...] Calculating the similarity matrix of the observation trace can only help the attacker decide the phase classification of his own observation, not any information about the victim's execution traces."

SVF uses phase correlation – which measures correlation in attacker's observed phases to phases in victim execution, a form of information leakage – for two important reasons. First, phase correlation likely reveals more leakage than direct correlation as direct correlation can only pick up on leakage which results in linear correspondence between victim and attacker traces, whereas phase correlation can pick up on more complex patterns and thus non-linear correspondence. Quantitatively, our study in Section 4 (specifically the comparison of hashing schemes in direct and phase correlations) provides some evidence that in practice phase correlation finds more leaks than direct correlation.

The second reason phase correlation is important is that it provides more flexibility in selecting trace points. Simply put, direct correlation can only compare trace points with similar events (*e.g.,* scalars vs. scalars or N-length vectors vs. N-length vectors). Phase correlation, however, can be used to compare any trace points for which similarity functions can be defined, a far wider range than direct correlation. Further, our results demonstrates the importance of trace selection in measuring multi-component leakage.

While none of this *proves* that phase correlation is entirely appropriate – in fact the SVF paper lists several occasions in which it may be ineffective – the results in this paper provide some evidence of its utility in designing metrics (which we show to be important) and efficacy in finding leaks. Our results should only be interpreted to support the utility of SVF – they apply only to the system which we are simulating and cannot be generalized.

5.2 On Characteristics of Metrics and Ground Truths

In the CSV paper's Section 2.2. "Characteristics of a Good System Vulnerability Metric", the authors lay out six characteristics of good metrics: realistic, deterministic, consistent, unbiased, instructive, universal. We agree with several of them: metrics should indeed be unbiased and deterministic. Ideally, they would also be universal and instructive, but these two may not be possible in all scenarios, as described in both Demme et al.'s and Zhang et al.'s papers. Zhang et al. also suggest (Section 2.2, Bullet 1) that metrics should be:

> "Realistic. It must correctly represent reality, e.g. the vulnerability of a system, and must represent 'ground truth'."

Resulting from the realism desiderata, the CSV paper validates their results (and refutes SVF results) by drawing on "known system behavior" which they term as "ground truths". The CSV paper does not define how system behavior is known or to whom. Their statements about "ground truths", however, seem reasonable for a cache-only context, and indeed our results verify this. On the other hand, as our experiments show, their "ground truths" do not necessarily hold in a multi-component context, which the original SVF paper examined. While we agree that metrics should be realistic, "ground-truths" must be precisely defined before they can used to judge metrics. In the next section we define "ground truth" and discuss challenges in determining ground-truth.

[3] Reference [9] in the CSV paper is a citation to the original SVF paper

6 Side Channel Metrics: Open Problems

As software is gradually hardened over the next several years (due to heavy investment in hardware and software for security), side-channels will become a major threat if unattended. Further, greater sharing of resources (vis-a-vis movement to the cloud) is likely to increase exposure to side-channel attacks. As such, methods to quantitatively understand leakage will become increasingly important; however, it will likely remain difficult to prove the relevance of leakage metrics and measurement methods. Thus questioning and critically analyzing metrics will remain an important topic.

Many open problems remain in the measurement of side-channels leaks. In addition to demonstrating the relevance of measurement methods and analyses like phase correlation, the selection of appropriate attack models and traces appear important yet are lacking in systematic methodologies.

Both SVF and CSV model an attacker and measure the leakage which that particular attacker can observe. As a result, the leakage which they measure is with respect to a particular attacker. Is it possible for real attackers to be more clever than the attacker models used by the metrics and find bigger leaks? Our experiments regarding asynchronous versus synchronous attackers – which are reasonably similar as they both use prime-and-probe cache scanning – indicate that the attacker model can have a strong effect on leakage. Radically different attackers are likely to probe systems in different manners and thus evoke different side-channel leaks. This begs the question: can systems' leakages be modeled independently from an attack model or are they two inherently intertwined? If the latter, do there exist systematic methods to find effective attack models which cover the full space of attacks? These questions are fundamental to side-channel measurement.

The selection of traces is a related problem. Using victim and attacker traces which directly correspond to an attack's target data and actual observations is a natural way to define traces, but it ties the measurement closely to a particular attack model. Can traces be defined differently to create measurements which are less dependent of the attack model? One could systematically examine many (or all) different "probe points" in a system to find leaks; however, would these leaks be located without also attempting all possible attackers? If the system can be exercised broadly and generically, perhaps systematic trace selection could locate leaks without thorough attack modeling.

Another interesting open question is determination of "ground truth". Before "ground truths" can be used to validate metrics they must be defined. In our view "ground truth" must be defined as a fact which has been rigorously proven within the same context and using the same set of assumptions as the metric. For example, it may be possible to mathematically prove that some cache features like static partitioning entirely prevent leakage if cache line contention alone is considered to be leakage. In this case, any valid metric for leakage should indeed show zero leakage, but only if the metric also considers cache line contention as the sole source of leakage. If the metric can be affected by any other factor – as both SVF and CSV can be – then the measurement and the mathematical proof can reasonably disagree. One reason for disagreement could be that the mathematical proof is using insufficient or simple axioms about the context in which leaks are occurring (and hence cannot be a "ground truth"). Improving the accuracy of the mathematical model, however, is extremely challenging as modern processors use complex features that often defy axiomatization and formal reasoning. Accordingly useful mathematical ground-truths rarely exist today; determination of these ground-truths is a challenging area of study.

7 Conclusions

In this paper, we have reviewed side-channels in general, systematically analyzed the differences between SVF and CSV, and rebutted many of the claims made by Zhang et al. [26]. These discussions have left us with two important conclusions regarding SVF vs. CSV and the use of "ground truths".

SVF vs. CSF We posed two important questions in the introduction: (1) Is CSV a sufficient metric? (2) Which of the differences between the two are most relevant? Our deconstruction in Section 4 indicates that SVF's flexibility to examine leakage through multiple components is important to understand the leakage of multi-component ensembles rather than individual components. This difference is likely the most important finding in this paper. Additionally, in the reduced scope of caches we presented some evidence that the limited analytical capability of direct correlation (versus SVF's phase correlation) can underestimate leakage in some scenarios. The flexibility provided by SVF combined with its ability to estimate more leakage is likely to make SVF more useful to processor designers.

Ground Truths We discussed the use of "ground truths" for metric validation in side-channel research. Mathematically proven ground-truths in a full-system context are difficult to obtain for modern processors because of their complexity. Alternately, using intuition as ground truth for metric validation is a dangerous concept – measurements should inform intuition rather than vice versa. Finally, "first-order" models of processor operation are also insufficient for side-channel security evaluation for the simple reason that the study of information leakage is a very young field, so we do not yet fully understand its dynamics to know which interactions and components within a processor can be ignored.

The paucity of full-system mathematical ground-truths, the danger of using intuition, and unsuitability of first-order estimates means that we will likely rely on full-system empirical measurements to measure leakage. Accordingly, our second conclusion is that metrics and methods should be continuously questioned and debated, but measurements disagreeing with intuition or a mathematical proof based on contrived axioms should be no more than a launching pad for further inquiry. In the context of security, any metric or framework that reveals unintuitive or surprising behaviors is an asset.

Acknowledgments

We thank Dr. Ruby Lee and Dr. Zhenghong Wang for clarifying the precise details of RPCache [23] while we implemented it during June/July of 2012. We also thank anonymous reviewers, and members of the Computer Architecture and Security Technologies Lab (CASTL) at Columbia University for their feedback on this work. This work was supported by grants FA 865011C7190 (DARPA), FA 87501020253 (DARPA), CCF/TC 1054844 (NSF), C-FAR/SRC/MARCO/DARPA contract number 2013-MA-2384, and gifts from Microsoft Research, WindRiver Corp, Xilinx and Synopsys Inc. This work is also supported through an Alfred P. Sloan Foundation Fellowship. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government or commercial entities.

8 References

[1] Onur Aciiçmez. Yet another microarchitectural attack:: exploiting i-cache. In *Proceedings of the 2007 ACM workshop on Computer security architecture*, pages 11–18. ACM, 2007.

[2] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In *Topics in Cryptology–CT-RSA 2007*, pages 225–242. Springer, 2006.

[3] Onur Aciicmez and J-P Seifert. Cheap hardware parallelism implies cheap security. In *Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on*, pages 80–91. IEEE, 2007.

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[5] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 191 –206, may 2010.

[6] 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.

[7] Leonid Domnitser, Nael Abu-Ghazaleh, and Dmitry Ponomarev. A predictive model for cache-based side channels in multicore and multithreaded microprocessors. In *Proceedings of the 5th international conference on Mathematical methods, models and architectures for computer network security*, MMM-ACNS'10, pages 70–85, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):35, 2012.

[9] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games – bringing access-based cache attacks on aes to practice. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 490–505, May 2011.

[10] Ralf Hund, Carsten Willems, and Thorsten Holz. Practical timing side channel attacks against kernel space aslr. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 191–205. IEEE, 2013.

[11] Suman Jana and Vitaly Shmatikov. Memento: Learning secrets from process footprints. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 143–157. IEEE, 2012.

[12] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. pages 388–397. Springer-Verlag, 1999.

[13] Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 286–296, New York, NY, USA, 2007. ACM.

[14] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 118–129. IEEE Computer Society, 2012.

[15] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of power analysis attacks on smartcards. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.

[16] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *CT-RSA*, pages 1–20, 2006.

[17] Colin Percival. Cache missing for fun and profit, 2005.

[18] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *Micro, IEEE*, 23(6):84 – 93, nov.-dec. 2003.

[19] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. Timing channel protection for memory controllers. In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014.

[20] Yao Wang and G Edward Suh. Efficient timing channel protection for on-chip networks. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 142–151. IEEE, 2012.

[21] Zhenghong Wang and R.B. Lee. A novel cache architecture with enhanced performance and security. In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pages 83 –93, nov. 2008.

[22] Zhenghong Wang and Ruby B. Lee. Covert and side channels due to processor architecture. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, ACSAC '06, pages 473–482, Washington, DC, USA, 2006. IEEE Computer Society.

[23] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. *SIGARCH Comput. Archit. News*, 35:494–505, June 2007.

[24] Hassan MG Wassel, Ying Gao, Jason K Oberg, Ted Huffmire, Ryan Kastner, Frederic T Chong, and Timothy Sherwood. Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 583–594. ACM, 2013.

[25] Andrew M White, Austin R Matthews, Kevin Z Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 3–18. IEEE, 2011.

[26] 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*, page 2. ACM, 2013.

[27] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 313–328. IEEE, 2011.

[28] 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*, pages 305–316. ACM, 2012.