Side Channel Vulnerability Metrics: the Promise and the Pitfalls

Tianwei Zhang
Princeton University
tianweiz@princeton.edu

Si Chen
Princeton University
sichen@princeton.edu

Fangfei Liu
Princeton University
fangfeil@princeton.edu

Ruby B. Lee
Princeton University
rblee@princeton.edu

ABSTRACT

Side-channels enable attackers to break a cipher by exploiting observable information from the cipher program's execution to infer its secret key. While some defenses have been proposed to protect information leakage due to certain side channels, the effectiveness of these defenses have mostly been given only qualitative analysis by their authors. It is desirable to have a general quantitative method and metric to evaluate a system's vulnerability to side-channel attacks.

In this paper, we define the features of a good side-channel leakage metric. We review a recently proposed metric called the Side-channel Vulnerability Factor (SVF) and discuss its merits and issues. We suggest the CSV metric, which tries to show how to overcome some of the shortcomings of the SVF metric, without completely changing its character. We use software cache side-channel attacks and defenses as an example to compare the metrics with known and measurable results on system leakiness.

1. INTRODUCTION

The confidentiality of secret or sensitive information is often protected by encryption.The encryption key is kept secret and accessible only by authorized parties. If the secret encryption key is leaked, then the confidentiality protection provided by strong encryption is nullified. Cryptographic algorithms are often designed to make it hard to recover the secret key through mathematical analysis.

Different methods are proposed to break the cipher. The simplest brute-force attack requires exponential time and energy - hence, if the key is long enough, it would be computationally infeasible to try all possible combinations of the key. However, in practice, side-channel attacks enable attackers to break the cipher in a much shorter time, with much fewer trails.

HASP'13, June 23 - 24 2013, Tel-Aviv, Israel

Unlike normal communication channels like buses and interconnection networks, side channels are not intended for communications. As a computer operates, it gives off certain observable characteristics that may leak secret information, such as power measurements, heat, sound, electromagnetic radiation or timing measurements. For instance, the characteristics are different when a public-key encryption algorithm is processing a key bit of "1" versus a key bit of "0". Hence, the key can be recovered by taking certain measurements and then doing off-line analysis using statistical and machine-learning techniques.

The question we ask in this paper is whether certain microarchitectural features make the system more vulnerable to side channel attacks? Are there certain design parameters that make a system more susceptible to side-channel leaks? It is highly desirable to have a metric that can indicate a system's vulnerability to side channel attacks. This could guide computer architects in making design decisions and security, performance and power tradeoffs.

Recently, a metric called the Side-channel Vulnerability Factor (SVF) was proposed [9] that tries to measure a system's vulnerability to all side channel leaks. While promising as a new hardware vulnerability metric, we find that there are problems with its scope, definition and measurement. Although a good metric is desirable, a defective metric may be misleading, resulting in security designs that do not actually improve security while needlessly degrading performance. The search for a more reliable metric motivates us to discuss and improve upon the SVF metric, without completely changing its character, in this paper. Our improved CSV metric gives a more meaningful measure by appropriately restricting the scope (albeit to some of the most important classes of microarchitectural side-channel attacks) and giving a definition that is consistent with known system behavior(sometimes referred to as "ground truth").

The contributions of this paper are:

- Providing an understanding of what side-channels are, and the requirements of a good metric to determine a system's vulnerability to side channel attacks.
- Identifying the issues with the SVF metric proposed by Demne et al [9].
- Making improvements to overcome these problems that helps designers understand what information leakiness

is and how to mitigate it.

The rest of the paper is organized as follows: Section 2 gives the background of various side-channel attacks, as well as the SVF metric proposed in [9]. We also look at the cache side-channel attacks as a case study, as in [9]. Section 3 details the limitations of SVF and the inconsistency of its results. In section 4, we introduce our improvements in the CSV metric and report its results. Section 5 concludes.

2. BACKGROUND

2.1 Side Channel Attacks

Side-channel attacks aim to break a cipher by exploiting the side-channel information. This by-product information can be retrieved from the implementation of the cryptographic algorithms in a physical system. Researchers have discovered a variety of side-channel information existing in a computation system, like a program's execution time [11], electromagnetic radiation from the device [10], power consumption [12], fault detection [7], etc.

There are two factors which determine if the side-channel attacks can be performed successfully [13]: from the system's aspect, the system must leak some useful information to the attacker; from the attacker's aspect, the attacker must be able to capture this information and recover keys or protected data based on on-line or off-line analysis. As a result, *the system's vulnerability to side-channel attacks includes both the system's vulnerability to leak the information and the attacker's capability to recover the information.*

2.2 Charateristics of a Good System Vulnerability Metric

As a metric is highly desirable to measure a system's vulnerability to side-channel attacks, we first consider the characteristics of a good metric. We list these below:

- **Realistic.** It must correctly represent reality, e.g. the vulnerability of a system, and must represent "ground truth".
- **Deterministic.** When given the same inputs, it must give the same results each time.
- **Consistent.** It should show the same trend in different systems. It must give correct comparisons of systems. $Metric(A) > Metric(B)$ means system A is more vulnerable than system B. Conversely, if A is more vulnerable than B, then Metric(A)>Metric(B).
- **Unbiased.** It must not be biased or show preference for some systems over others.
- **Instructive.** It should help computer architects to understand which features or parameters are more prone to leaking critical information under which types of attacks, and guide them to make intelligent design decisions.
- **Universal.** It should be applicable to all computer systems for all kinds of side-channel attacks.

Recently Demme et al. defined a metric to evaluate a system's vulnerability to side-channel information leakage [9]. We summarize and discuss their metric and results in this paper.

2.3 Side Channel Vulnerability Factor, SVF

The Side-Channel Vulnerability Factor(SVF), introduced in [9], is a metric to measure the correlation between a victim's execution and an attacker's observations. First, the victim's execution trace and the attacker's observation trace are collected, and patterns in both traces are detected using phase detection techniques. Then, the correlation between the actual patterns in the victim trace and the observed patterns are computed. We give a condensed mathematical definition of the SVF metric below:

Let $v_1, v_2, \ldots, v_n$ be the victim's execution trace (termed the oracle trace in [9]) and $a_1, a_2, \ldots, a_n$ be the attacker's observation trace (or the side-channel trace). Suppose there are k measurement results for each trace at time interval i, we can denote the two traces as

$$\vec{v}_i = [v_i(1), \ldots, v_i(k)]; \quad \vec{a}_i = [a_i(1), \ldots, a_i(k)] \tag{1}$$

Demme uses the Euclidean distance to define the victim's similarity matrix M_v as:

$$M_v(i,j) = \begin{cases} D(\vec{v}_i, \vec{v}_j), & \text{if } i > j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $D(\vec{v}_i, \vec{v}_j) = \sqrt{(v_i(1) - v_j(1))^2 + \ldots + (v_i(k) - v_j(k))^2}$. The same definition is made to get the attacker's similarity matrix M_a.

Finally, Demme defines the SVF as the Pearson correlation coefficient of vectorized forms of M_v and M_a:

$$SVF = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(M_a(i,j) - \overline{M_a})(M_v(i,j) - \overline{M_v})}{\sigma_{M_a}\sigma_{M_v}} \tag{3}$$

where $\overline{M_a}$, $\overline{M_v}$ and σ_{M_a}, σ_{M_v} are the average and standard deviation of attacker's and victim's similarity matrices.

2.4 Case Study:Cache Side-Channel Attacks

Information can be leaked out through side channels because different programs can share the same hardware, such as processors and memory. So caches [14, 2], branch predictors [3], pipelines [5, 16] and other shared resources are all potential sources of information leakage. Among all the side-channel attacks, cache side-channel attacks are particularly dangerous. Since caches are one of the most important performance features in a computer, they are almost universally present in all processors, from embedded processors to high-end multi-core processors for Cloud servers. Furthermore, many optimized software implementations of cryptographic algorithms use table lookups. Since the tables are stored in memory, caches are implicitly involved in the execution of these cryptographic programs. Also, cache side-channel attacks do not require extra equipment, hence they are easy to perform.

In cache-based side-channel attacks, the adversary does not need to access the victim's bus or memory. The attacker process and the victim process can be isolated, and they do not need to share the same address space. However, they may still share the hardware caches, thus providing the attacker a side-channel to observe the victim's secret information. Based on the actions the attacker can perform and the information he can observe, the attacks are often divided into three types: access-based attacks [15], timing-based attacks [6] [8] and trace-based attacks [4].

One of the common techniques for the attacker to steal information from the victim is the "Prime and Probe" attack [15] [14]. In this method, the attacker allocates a contiguous byte array, the size of which is equal to the cache size. At the first "Prime" stage, he will read a value from every memory block in the array. This will evict all the victim's lines. At the second "Probe" stage, he will again read each block in the array, and measure the time of each memory access. A large time means a cache miss, which indicates that this cache set has been accessed by the victim between the "Prime" and "Probe" stages. By inferring the state of the cache for the victim's encryption program between the "Prime" and "Probe" stages, the attacker can obtain the cache accesses (and thus the memory accesses) made by the victim's program during that interval. The attacker can analyze such side-channel information to recover confidential data. The Probe stage is the Prime stage for the next interval.

3. DISCUSSIONS ON SVF

Demme's SVF [9] provides a nice starting point for finding a metric to measure the leakiness of a system; at the same time, it also has some limitations, which we discuss below. This is important, as computer architects may otherwise be misled into making incorrect design decisions.

3.1 Scope

The aim of SVF is to use a single metric to reveal the information leakage of the entire system for all side-channel attacks. This is highly desirable, but may also be too hard to realize. Security experts would typically agree that there does not exist a universal way to measure vulnerability, considering there are so many side-channels and different types of attacks, known or unknown. 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. Just as Demme et al. admitted that *SVF cannot evaluate unknown side-channels, the scope of SVF in [9] may be a bit overstated.*

Let's take the case of cache-based side-channels as an example. In [9] they measure the number of accesses to each cache set as the victim's trace, and the load time of each set in the probe stage as the attacker's trace. Wang and Lee [17] identify the root cause of cache side-channel attacks as interference (external or internal). So we think calculating this SVF is only applicable to cache side-channel attacks caused by external interference, more specifically, access-based attacks using the "Prime and Probe" attack. Attacks due to internal interference, like timing-based attacks, cannot be evaluated by this SVF since there is only aggregated time that can be measured, but no trace of an attacker's observation (in a time series). So *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 suggest reducing the scope to a class of cache side-channel attacks.*

3.2 Definition

Similarity Matrix: In [9], the Euclidean distance is used to form the similarity matrix. Similarity matrix is a great tool to compare the similarity between different points in one trace, but it may not be appropriate to calculate the correlation of two similarity matrices to compare their traces. The side-channel attacker who performs access-based attacks on the cache tries to infer the victim's execution trace from his observation trace. 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. *We suggest the following improvements to the SVF metric: calculate the correlation coefficient directly between the attacker's observation traces and the victim's execution traces without the similarity matrix. Calculating the correlation of their similarity matrices is unnecessary and may give wrong results since the Euclidean distance removes information of individual elements in the trace.*

Traces: Consider the values of traces collected for an SVF calculation. At the attacker's side, the trace is the cache access time; at the victim's side, the trace is the number of accesses. However, we think the two traces are not correlated. The attacker can only know if one cache set is accessed by the victim, but he does not know the *number* of accesses. It is possible that one cache block is accessed by the victim many times during one interval, but only one cache miss is observed by the attacker since only one attacker's block is evicted by the victim. *We suggest the following improvements to the SVF metric: use binary values to represent the two traces. For the attacker's traces, "1" means for each cache set, there is at least one cache miss and "0" means all lines are cache hits; for the victim's traces, "1" means there is at least one cache access and "0" means no cache access at all, for each set. This will make the correlation between the attacker's and the victim's traces less noisy.*

3.3 Measurements

System's vulnerability vs Attacker's capability: Since side-channel attacks are determined by the system's vulnerability as well as the attacker's abilities, defining a metric of vulnerability to side-channel attacks should consider both of them. At the system's side, since not all the information revealed by a system is critical, only the critical information should be considered to study the information leakage of the system. Observing non-confidential information of the system is not called "side-channel information leakage". SVF can not distinguish the differences between public and secret information of the system, which may lead to some misleading conclusions.

At the attacker's side, since a good metric will reveal the system's vulnerability to side-channel attacks, it should be decoupled from the attacker's capability to retrieve information. The best way to realize this is to maximize the attacker's capability. If the attacker's capability is too low, even if the system is very vulnerable to side-channel attacks, the attacker can not get much information. So calculating such a metric may lead to wrong conclusions.

Let's look at the cache case study in [9]. For access-based attacks, the attacker's capabilities depend on two aspects: the way it performs the "Prime and Probe" attacks and the interval it chooses.

There are two main ways to perform access-based attacks: synchronous and asynchronous attacks [14]. In synchronous attacks, the attacker has some interaction with the victim program. He is allowed (e.g., scheduled) to execute the "Prime" and "Probe" synchronously before and after the victim's program. In asynchronous attacks, the attacker does not have explicit interactions with the victim's program. He only needs to run the "Prime and Probe" process in the

same processor as the victim's encryption process. Clearly synchronous attacks require more attacker capability.

Another important issue is the selection of intervals. In a "Prime and Probe" attack, the interval granularity reflects the attacker's capabilities to retrieve information from the system. The smaller the granularity, the higher the attacker's capability, and the more information he can retrieve. A rather large interval, during which almost every cache set is accessed by the victim, will prevent the attacker from retrieving useful information and make the "Prime and Probe" attack ineffective. The ideal granularity for the attacker in a "Prime and Probe" attack is to capture every access of the victim to memory. However, there is a limitation on the interval granularity for asynchronous attacks. It is limited by how fast an attacker can do the scan of the entire cache. For synchronous attacks, the interval can be made small enough to observe the victim's accesses accurately, depending only upon the granularity that the attacker's process can be scheduled alternately with the victim's process.

Demme et al. do not define the capability of an attacker, or give reasonable explanations for choosing the size of intervals and the method for performing the "Prime and Probe" attack. These may result in mixing up the attacker's capability and the system's vulnerability in an unpredictable way. We propose synchronous attacks and small time intervals, which maximize the attacker's capabilities.

3.4 Results

To better illustrate the SVF metric and its potentially misleading and inconsistent results, we select some configurations representative of real machines in table 1. All cache parameters are kept the same, except one of the 3 rows at the bottom of the table, which correspond to parameters we study. The SVF values from [9] are shown in figure 1, and discussed below.

Partitioning Policy: Demme discusses two kinds of partitioning policies. One is static partitioning, where two simultaneous processes occupy equal separate parts of the cache set-associativity ways in each set. So the victim and attacker actually do not share any cache lines. When the attacker probes the cache blocks that it has previously filled, it is always a cache hit. So the variance of the attacker's traces should be close to zero, making the Pearson correlation coefficient close to zero. The SVF for this static partitioning policy should ideally be 0, to represent "ground truth".

Another solution is dynamic partitioning. This divides the cache equally at the beginning, and each process gets half of the cache ways in each set. Every 10^6 cycles, the cache sets will be reallocated to the attacker and the victim, 75% of the ways of each set will be given to the program which uses the cache more frequently during the last 10^6 cycles, and the remaining 25% of the ways of each set will be given to the other program. Compared to static partitioning, dynamic partitioning may increase the performance, but it may leak some information during the cache way reallocation since it allows some amount of sharing. Considering the reallocation period is much larger than the "Prime and Probe" interval, most of the time the attacker program is isolated from the victim program and the information he will gather during way reallocation is rather small. Based on this analysis, it is easy to see that the SVFs of these methods should have the following relationship if the relative SVFs are representative of "ground truth":

$$SVF_{none} \geq SVF_{dynamic} \geq SVF_{static} \tag{4}$$

We study the SVF results with different cache partitioning policies. Here we fix all the cache parameters in table 1 (with no eviction and low-bit indexing is used). We vary only the partitioning policy: no partitioning, dynamic partitioning, and static partitioning. The results in [9] are shown in figure 1(a). From these results, it is very hard to see that static and dynamic cache partitioning have advantages in preventing information leakage, like equation 4 shows. In particular, for configuration **A** (32 Kbyte cache, 8-way set-associative (SA), 64 byte lines), static partitioning has a much higher SVF than no partitioning - which contradicts "ground truth". Also, the trends for the 4 configurations are not consistent (they should all 4 be decreasing in SVF values, from left to right). For a given cache size and associativity, the SVF results indicate that longer line size (64 bytes) is less vulnerable than shorter line size (8 bytes) for 8-way SA caches (except for static mapping), but the opposite is shown for 4-way SA caches. These contradictory results are not very useful to cache designers.

Eviction Policy: Here, random evictions are studied. A random cache line is evicted periodically and fosters the illusion that this cache line is accessed by the victim. It brings difficulties to the attacker for key-recovery as the attacker cannot differentiate the victim program's real memory accesses or the system's random evictions. From SVF's point of view, the introduction of random evictions should make the attacker's observed traces less correlated to the victim's traces, thus reducing the value of SVF. It is obvious that the more frequent a random cache line is evicted, the more noise is added into the attacker's traces, thus the lower the SVF should be. In [9], two eviction randomizations are studied: *100% Eviction* evicts a cache line every cycle and *50% Eviction* evicts a cache line every cycle with 50% probability. At such a high frequency, a large amount of "fake" lines are evicted between the attacker's two consecutive accesses to the cache. So the SVF should be much smaller than a conventional cache with no random evictions. The relationship expected is that the SVF with no evictions is greater than that with random evictions, although we cannot be sure of the relationship between 50% and 100% evictions. So we have:

$$SVF_{none} \geq \{SVF_{50\%}, SVF_{100\%}\} \tag{5}$$

We show the results in figure 1(b) for these three different eviction policies. We use the cache parameters in table 1(with no partitioning and low-bit indexing). For configuration **D**, 50% eviction has higher SVF than no eviction, and 100% eviction has the highest SVF, which is inconsistent with equation 5. In addition, we see inconsistent effects of cache line size on SVF for 4-way and 8-way SA caches, as in figure 1(a) for partitioning.

Indexing Policy: Conventional caches use the low bits of an address to index the cache set. Another policy is calculating the bitwise XOR of two parts of the memory bits to index the cache. Theoretically, these two methods should carry the same amount of information. Since the memory-to-cache mappings are fixed and known to the public, the attacker can easily deduce the original memory address from the cache set index observed. The indexing policy can be effective in reducing information leakage when the mappings

between memory addresses and cache sets are randomized [17, 18]. Here we detail two random-mapping cache architectures:

RP (Random Permutation) Cache [17] uses randomization on cache misses to defend against cache side-channel attacks. Each process may have a *Permutation Table* (PT), which dynamically maps the memory address to a cache set. When a process's cache miss occurs and a victim cache line belonging to another process is chosen for replacement, instead of evicting and replacing this victim line, thus revealing information to outsiders, a new cache line in a new set is selected randomly, evicted and replaced by the incoming cache line. At the same time, this process's *Permutation Table* is updated by swapping the two corresponding sets, and the lines in these two sets belonging to this process need to be invalidated. The randomization and dynamic swapping mechanism can effectively reduce the external interference between different processes. Actually, RP Cache is not just an indexing method, but it also combines aspects of random eviction for protection against internal interference as well as the external interference just described. However, its random eviction is not like the 50% or 100% random evictions described above and in [9], which can cause large and unnecessary performance degradations.

PRS (Permutation Register Sets) Cache [9] uses the same idea of Permutation Tables as RP cache (calling them PRS instead), but it has a very different replacement algorithm. Instead of swapping the permutation table entries on cache misses, it swaps two randomly selected lines periodically. For instance, the PRS Cache selects two lines in the victim process's PRS every 100 loads and swaps them. Then after a certain time, all the lines are swapped many times and the permuted indices in the PRS become rather random. Since the attacker does not know the memory-to-cache mappings given by the victim's PRS, no information is leaked on memory to cache mappings.

Based on the analysis above, it is clear that caches with LOW bit indexing and with XOR indexing should have the same vulnerability to side-channel leakage, and higher than RP cache and PRS Cache. So we have:

$$SVF_{low} \approx SVF_{xor} \geq \{SVF_{prs}, SVF_{rp}\} \tag{6}$$

We now study the results for these different cache indexing policies. Figure 1(c) shows the SVF results from [9] for the first three indexing mappings. (Actually in [9] Demme et al. only gave the data of PRS, which they liken to RP cache. Post [9] publication, they generated the SVF data for RP cache.) In configurations **A** and **C** (8-way versus 4-way SA caches, both with 64 byte lines), PRS cache has unusually high SVFs, much higher than no indexing or any of the other indexing schemes. This has led some people to conclude that using security features like re-mapping registers (introduced by RP cache) produces caches that are in fact less secure than conventional caches. This is misleading and incorrect. The reason for the high SVF for PRS cache given in [9] is that it might create some unknown timing and pipeline channels – which is unsubstantiated and also contradictory to their earlier claim that SVF cannot measure unknown side-channels (as we discussed in section 3.1). Rather, these inconsistent SVF results may be more likely due to noise in both the SVF definition and in its measurements. The inconsistency in the results is further seen in configurations B and D (8-way and 4-way SA caches, both with 8 byte lines), where now the SVF values are about the same for all the caches.

Based on the above results, this SVF metric does not satisfy the requirements of our definition of a good metric for being Realistic, Consistent and Instructive.

4. CSV: IMPROVEMENTS TO SVF

In this section, we suggest some improvements to Demme's SVF. We call the new metric the **Cache Side-channel Vulnerability (CSV)** metric. We use the gem5 simulator [1] to simulate the different cache configurations and we calculate CSV values. Our results show that CSV is more reasonable than SVF [9] in that it reflects cache side-channel leakage more accurately. Some key insights are to limit the scope appropriately, so that the metric can actually reveal which cache design parameters are more likely to leak than alternative ones, hence making the improved CSV more instructive for making system design choices for security. We also reveal system vulnerabilities more accurately by not mixing it up with the attacker's capabilities; we do this by assuming the most powerful attacker in all cases.

4.1 SVF Improvements

Scope: We define the scope of CSV to reveal the system vulnerability to cache side-channel information leakage only, and by access-based attacks only. Other kinds of side-channel attacks, e.g. pipeline side-channel attacks, or cache-based timing-based attacks, are not properly reflected. We believe this may also be the true reduced scope of the SVF defined in [9].

Definition: We define the CSV as follows: Let $v_1, \ldots, v_n$ be the victim's execution trace and $a_1, \ldots, a_n$ be the attacker's observation trace. If the number of cache sets is k, then

$$\vec{v}_i = [v_i(1), \ldots, v_i(k)]; \quad \vec{a}_i = [a_i(1), \ldots, a_i(k)] \tag{7}$$

where $v_i(j)$ is 1 if the victim accesses set j at time interval i, or 0 if it does not; and $a_i(j)$ is 1 if the attacker's access time of set j shows a cache miss, or 0 if it is a cache hit. Comparing with the SVF definition in [9], we use binary values in both the victim's and attacker's traces instead of the number of misses and the load time, respectively, for each cache set.

We also abandon the use of Euclidean distance and the Similarity Matrix, and simply put these vectors together to form new matrices for the victim and attacker. So

$$M_v(i,j) = v_i(j); \quad M_a(i,j) = a_i(j) \tag{8}$$

The CSV is the Pearson correlation coefficient of these two matrices as in equation 3.

Measurements: The vulnerability to side channel leakage includes both the system's intrinsic information leakage as well as the attacker's capabilities to observe information. To evaluate a system's vulnerabilities, we should filter out the attacker's capabilities. One option is to always maximize the attacker's capabilities. In [9], Demme et al. adopt the asynchronous attacker model and 10,000 victim instructions as the interval granularity, but they do not give a clear rationale for this choice. In our improved metric, to increase the attacker's capabilities, we run *synchronous* "Prime and Probe" attacks with a much smaller interval granularity, in which the attacker's and victim's programs will run in turn. The attacker will first initialize the cache, filling up all the

lines during the "Prime" stage. After the victim's program executes for a very small interval, the attacker will probe the cache, and at the same time prime the cache for the next interval.

For the size of the interval, although we prefer smaller granularities, a small interval may not reveal some cache security protections, e.g., when the memory system uses non-blocking cache technology. Even though there is a cache miss for the victim, the accessed line will be stored in the MSHR (Miss Status Handling Register) in the gem5 simulator, for example, and the following instruction will be executed before the missed cache line is fetched from memory to cache. This not only reduces the cache miss penalty, but also adds some noise in the cache access trace, since the victim's access may not give the attacker immediate notification. So if the interval is too small, the non-blocking cache effect will overshadow the other security countermeasures we care about, making it very hard to compare the effectiveness of these protections. In our experiments, we choose 100 cycles as the interval, which is fine for both the attacker retrieving information and for revealing the effects of security protections.

4.2 Implementation and Configurations

We use gem5, a research simulator developed and improved over many years to implement caches and measure our improved metric, rather than the less mature, home-grown simulator used in [9]. Gem5 is a modular platform encompassing system-level architecture as well as processor microarchitecture. We emulate the ARM ISA, system call emulation mode, with an out-of-order CPU, and 2GHz clock frequency. The cache parameters (same as in [9]) that we select are listed in table 1. We implement a three-level cache. The sizes of L1, L2 and L3 are 32KB, 256KB and 8MB. The associativity of L1, L2 and L3 are the same: either 8 way or 4 way. The cache line size in all the caches is also identical: either 64B or 8B. We run an AES program encrypting 100 blocks of random text as the victim's program.

Parameter	**Value**
Cache Size	L1=32K, L2=256KB, L3=8MB
Line Size	64B or 8B
Associativity	8-way or 4-way
Single or Multi core	Single
Cache Sharing	L1, L2 and L3
Prefetch	None
Attacker Style	In Order
Partitioning	No, Dynamic, Static
Eviction	No, 50%eviction, 100%eviction
Indexing	LOW, XOR, PRS, RP

Table 1: Cache configurations for figures 1-3. The parameters above the double line are fixed for all configurations while we vary one of the three policies below the double lines (the other two default to the first value listed).

4.3 Results

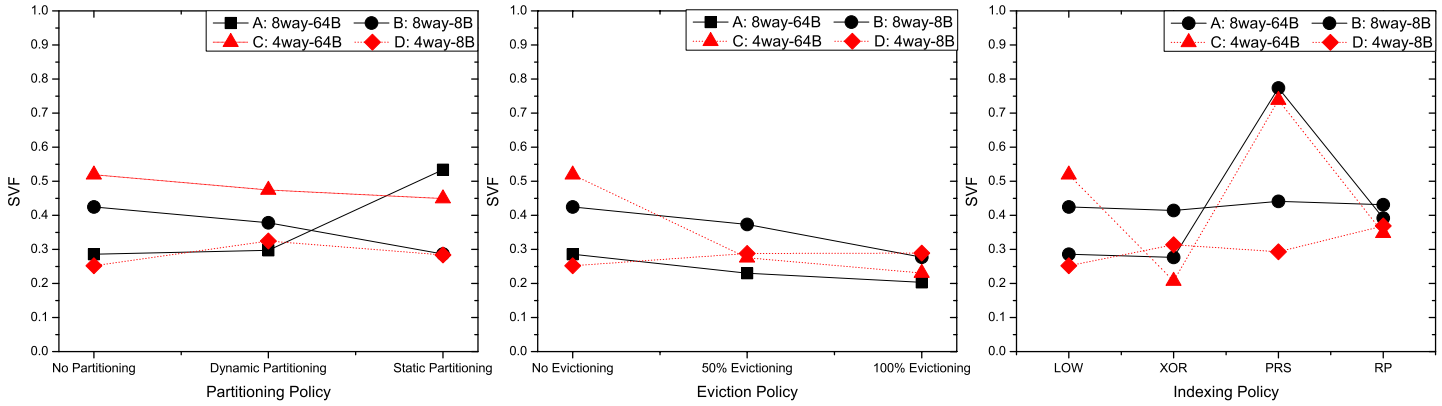
Improvement on Measurements: First, we improve SVF only on the measurements by adopting synchronous "Prime and Probe" techniques, and using the well-established gem5 simulator. The results are shown in figure 2. From figure 2(a) we can see that the currently measured SVF shows an SVF of zero for static partitioning, with a higher SVF for dynamic partitioning and the highest SVF for no partitioning. This agrees with reality as expected in equation 4. Also, these results are more consistent for all 4 configurations, unlike the original SVF values in 1(a). However, for eviction policies and indexing policies, the SVF results are almost the same for the different configurations. So they cannot properly show the system's vulnerability to side-channel leakage, and hence are not instructive to system designers.

Our improved CSV: We calculate our improved CSV as described in section 4.1. For measurements, we make an improvement by using synchronous "Prime and Probe" techniques; for definition, we make improvements by removing the similarity matrix and using binary traces. Our results are shown in figure 3 and analyzed below:

Partitioning Policy: We compare the same three kinds of partitioning policies as in [9]: no partitioning, static partitioning and dynamic partitioning. Our results in figure 3(a) show clearly that no-partition caches have the highest CSV values from 0.55 to 0.9. For the static partitioned caches, the CSV values are zero for all configurations; since the attacker cannot probe the victim's cache ways, it is always a cache hit for every line when the attacker probes the cache, thus retrieving no information. For dynamic partitioning, information may leak when reallocating cache lines. The value is greater than static partitioning, but definitely less than no partitioning. This agrees with equation 4. If we compare configurations **A** and **B** (or **C** and **D**), which have the same associativity but different cache line sizes, we can see that smaller line size is more vulnerable than larger line size. If we compare configurations **A** and **C** (or **B** and **D**), which have the same line size but different associativity, we see that associativity differences (between 4-way and 8-way) do not have much effect on the system's vulnerability.

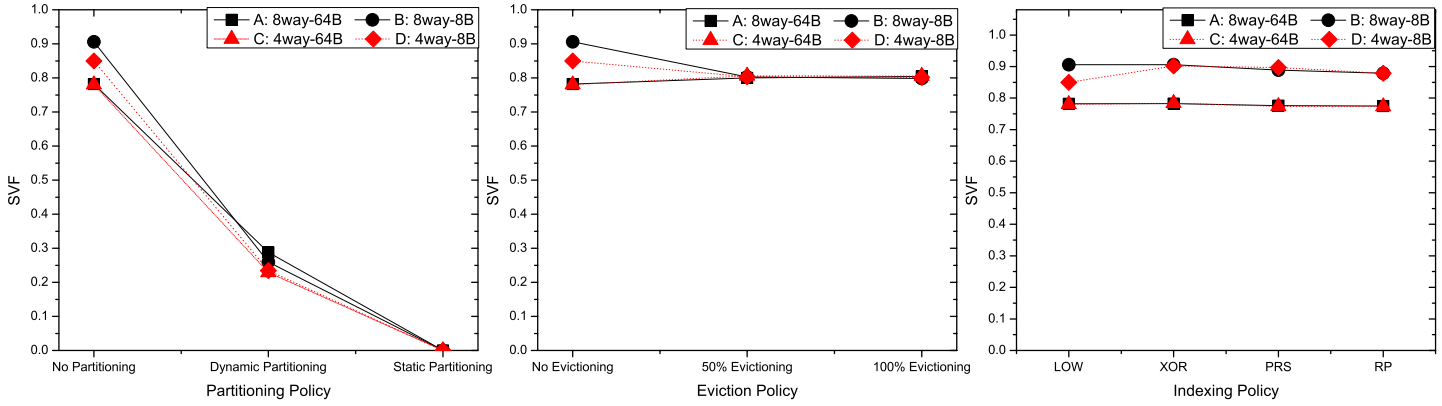
Eviction Policy: The second comparison is between no eviction, 50% random eviction and 100% random eviction. Theoretically, random evictions can add noise for the attacker's "Probe" stage, and thus have a lower SVF. Our result in figure 3(b) shows the CSV for no eviction is higher than with random eviction, and 100% eviction has slightly smaller CSV than 50% eviction as expected. This trend is consistent for all configurations, and agrees with equation 5. If we want to see the effects of line size and cache associativity to the system's vulnerability, we can get the same conclusions as the partitioning policy: smaller line size gives higher CSV vulnerability, but set-associativity of 4 or 8 has negligible impact.

Indexing Policy: We consider the same four indexing policies: using the low bits to index the cache (as in conventional caches today), bitwise XOR, the PRS cache and the RP cache. Our results in figure 3(c) show that the CSV of low bits and bitwise XOR are almost the same, for cache configurations with the same line size. Both RP cache and PRS cache have much lower CSV values than LOW and XOR indexed caches. So RP cache and PRS cache are effective designs to defend against the "Prime and Probe" and cache access-based side-channel attacks. From the figure, we can see RP cache is even less vulnerable to side-channel leakage than the PRS cache. If we swap two random lines more frequently, the CSV of PRS cache could be smaller. Consistent with figures 3(a) and (b), we see that larger line size is less leaky than smaller line size, and 4 or 8-way associativity has little impact on the CSV metric. Unlike the SVF results, these observations about line size and associativity are consistent in all three policies studied (for partitioning, eviction and indexing). A larger line size obscures



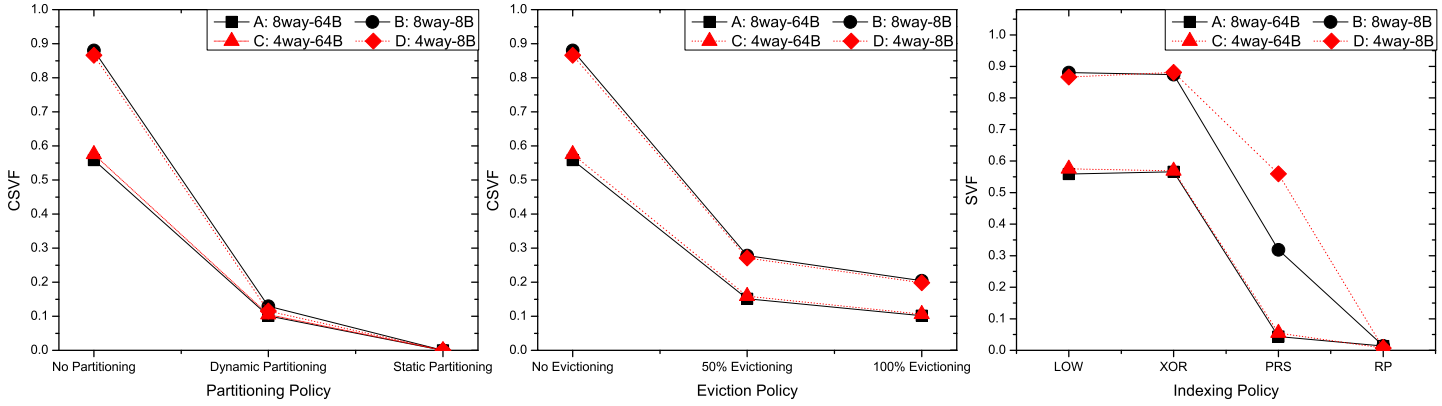
(a) SVF of Partitioning (b)SVF of Eviction (c) SVF of Indexing

Figure 1: Inconsistent results of SVFs from [9] for different configurations.



(a) SVF of Partitioning (b)SVF of Eviction (c) SVF of Indexing

Figure 2: Results of SVFs for different configurations on gem5 using synchronous "Prime and Probe" techniques.



(a) Improved SVF of Partitioning (b)Improved SVF of Eviction (c) Improved SVF of Indexing

Figure 3: Consistent results of improved CSV for different configurations.

the particular AES table entry accessed by the victim, since it contains more table entries.

4.4 Comparing SVF and CSV

We believe our CSV metric satisfies most of the good metric properties we defined in section 2.2 and improves upon the SVF metric in [9]. Both SVF and CSV are deterministic and unbiased metrics, since both give the same value for the same inputs and are not designed to show preference for one system over another. However, SVF may not be representative of reality and does not show consistent trends across different systems (as shown in figure 1 and section 3), while our CSV metric is both representative of reality and consistent (as shown in figure 3 and section 4). The inconsistent results from the SVF metric make it less instructive to computer architects, whereas our CSV metric can give useful and consistent comparisons on which cache parameter values may decrease vulnerability to cache side channels. Both our CSV metric and the SVF are not universal metrics(despite SVF's claim to be one), but a universal metric may not be all that desirable since it may not be instructive: it cannot tell the computer architect which parameters of a subsystem's design improves or degrades that subsystem's vulnerability to the side-channel attacks that target it.

5. CONCLUSIONS

In this paper, we describe side channels in general, and cache side channels in particular. We discuss what a good side channel vulnerability metric should be. Then we evaluate the side-channel vulnerability metric, SVF, proposed in [9]. We point out some limitations of this SVF metric: its scope, definition and measurements. We discuss their experimental data. We overcome most of these limitations of the SVF metric, and give an improved CSV metric. We remove the ambiguity of the system's vulnerability and the attacker's capability by assuming the most powerful attackers. We reduce noise in the definition of the SVF metric, by removing the similarity matrix, directly correlating the victim and attacker traces, and representing these traces as binary values: the victim either accesses a set or not, rather than the number of accesses for each set, and the attacker either observes a cache hit or a cache miss for a set, rather than the time taken for accessing each set. We show that the results of our CSV metric is consistent with ground truth (realistic), shows consistent trends across cache configurations, and thus is also more instructive to cache designers.

6. ACKNOWLEDGMENTS

We thank S. Sethumadhavan and J. Demme for giving us access to their data in [9] for figure 1, and collecting further data on RP cache. This work was supported in part by DHS/AFRL FA8750-12-2-0295 and NSF CNS-1218817 and CCF-0917134

7. REFERENCES

[1] The gem5 Simulator System. http://www.gem5.org.

[2] O. Aciiçmez. Yet another microarchitectural attack: Exploiting i-cache. In *Proceedings of the ACM Workshop on Computer Security Architecture*, pages 11–18, 2007.

[3] O. Aciiçmez, c. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, pages 312–320, 2007.

[4] O. Aciicmez and C. K. Koc. Trace-driven cache attacks on aes (short paper). In *Proceedings of the International Conference on Information and Communications Security*, pages 112–121, 2006.

[5] O. Aciicmez and J.-P. Seifert. Cheap hardware parallelism implies cheap security. In *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 80–91, 2007.

[6] D. J. Bernstein. Cache-timing attacks on aes. Technical report, 2005.

[7] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of the International Cryptology Conference on Advances in Cryptology*, pages 513–525, 1997.

[8] J. Bonneau and I. Mironov. Cache-collision timing attacks against aes. cryptographic hardware and embedded systems. In *Lecture Notes in Computer Science series 4249*, pages 201–215, 2006.

[9] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan. Side-channel vulnerability factor: a metric for measuring information leakage. In *International Symposium on Computer Architecture*, pages 106 –117, June 2012.

[10] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the Design Automation Conference*, pages 753–760, 2004.

[11] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of the International Cryptology Conference on Advances in Cryptology*, pages 104–113, 1996.

[12] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of the International Cryptology Conference on Advances in Cryptology*, pages 388–397, 1999.

[13] B. Köpf and D. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the ACM conference on Computer and Communications Security*, pages 286–296, 2007.

[14] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of aes. In *Proceedings of the Cryptographers' Track at the RSA Conference on Topics in Cryptology*, pages 1–20, 2006.

[15] C. Percival. Cache missing for fun and profit. In *Proc. of BSDCan 2005*, 2005.

[16] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *Proceedings of the Annual Computer Security Applications Conference*, pages 473–482, 2006.

[17] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the International Symposium on Computer Architecture*, pages 494–505, 2007.

[18] Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 83–93, 2008.