Comparative Measurement of Cache Configurations' Impacts on Cache Timing Side-Channel Attacks

Xiaodong Yu, Ya Xiao, Kirk W. Cameron, Danfeng (Daphne) Yao
Dept. of Computer Science
Virginia Tech, Blacksburg VA, 24060
{xdyu, yax99, cameron, danfeng}@vt.edu

Abstract

Time-driven and access-driven attacks are two dominant types of the timing-based cache side-channel attacks. Despite access-driven attacks are popular in recent years, investigating the time-driven attacks is still worth the effort. It is because, in contrast to the access-driven attacks, time-driven attacks are independent of the attackers' cache access privilege.

Although cache configurations can impact the time-driven attacks' performance, it is unclear how different cache parameters influence the attacks' success rates. This question remains open because it is extremely difficult to conduct comparative measurements. The difficulty comes from the unavailability of the configurable caches in existing CPU products.

In this paper, we utilize the GEM5 platform to measure the impacts of different cache parameters, including *Private Cache Size* and *Associativity*, *Shared Cache Size* and *Associativity*, *Cacheline Size*, *Replacement Policy*, and *Clusivity*. In order to make the time-driven attacks comparable, we define the *equivalent key length (EKL)* to describe the attacks' success rates. Key findings from the measurement results include (i) private cache has a key effect on the attacks' success rates; (ii) changing shared cache has a trivial effect on the success rates, but adding neighbor processes can make the effect significant; (iii) the Random replacement policy leads to the highest success rates while the LRU/LFU are the other way around; (iv) the exclusive policy makes the attacks harder to succeed compared to the inclusive policy. We finally leverage these findings to provide suggestions to the attackers and defenders as well as the future system designers.

1 Introduction

Timing-based cache side-channel attacks have been comprehensively studied since its debut in 1996 [17]. The time side-channel of cache leaks secret information, specifically through the time differences of cache hits and misses. Time-driven attacks are the primary type in the early age of the timing-based side-channel attacks. They observe the total execution time of the cryptographic operations and crack the keys by analyzing the observations.

In the last decade, access-driven attacks, for example, the notorious meltdown attack [18], have gained popularity. They manipulate specific cache-lines and observe the access behaviors of cryptographic operations on these cache-lines. They require fewer observations than the time-driven attacks due to the higher resolutions and lower noises. Fine-grained variants of the access-driven attacks have been proposed, including PRIME+PROBE [20], FLUSH+RELOAD [29], and FLUSH+FLUSH [14]. However, all access-driven attacks require that the attacker's process have access to specific cache addresses that shared by the victim's process.

Recently, some proposals leverage new hardware technologies to defend the access-driven attacks. For example, the CATalyst [19] exploits Intel's Cache Allocation Technology (CAT) (CAT isolates shared cache space for victim's process) to physically revoke the accessibility of attacker's process. Given the attackers' access privilege is not an assurance, studying time-driven attacks are still worthwhile since they demand no adversary's intervention during the observations.

Researchers have been speculating that cache configurations can impact the performances of time-driven side-channel attacks [4, 8, 9]. For example, intuitively, a bigger cache size would make the attacks on AES more difficult, as it can hold a larger portion of the precomputed S-box lookup table in the cache. However, we are not aware of any previous work that provides experimental data to demonstrate how cache configuration parameters influence time-driven attacks. It is extremely challenging to conduct performance comparisons under the same system with different cache configurations, as none of the existing CPU products provide configurable caches. This challenge prevents the experimental study about the impact of cache parameters on time-driven attacks.

In this work, we overcome the aforementioned difficulty and conduct a comprehensive study on how cache configurations impact the success rates of time-driven attacks. We leverage a modular platform – GEM5 [6] to measure the performances of time-driven attacks under various cache configurations. GEM5 is one of the most popular cycle-accurate full-system emulators in the computer-system architecture community [1, 30]. We leverage GEM5 to emulate the X86_64 system with a configurable cache.

In our work, we measure the performance of Bernstein's cache timing attack on AES [4]. Bernstein's attack is one of the most classic time-driven attacks and still feasible on model processors [2, 10]. To make its performance comparable, we propose a new metric to quantify the its success rate. In the measurement, we run Bernstein's attacks on GEM5 instances with different cache configurations and provide systematic experimental data to describe the correlation of cache parameters and the attack's performance. Our contribution can be summarized as follows:

- We use the GEM5 platform to investigate the cache configurations' impacts on time-driven cache side-channel attacks. We configure the GEM5's cache through seven parameters: *Private Cache Sizes*, *Private Cache Associativity*, *Shared Cache Sizes*, *Shared Cache Associativity*, *Cacheline Sizes*, *Replacement Policy*, and *Clusivity*.
- We extend the traditional success-fail binary metric to make the cache timing side-channel attacks' performances comparable. We define the *equivalent key length (EKL)* to describe the success rates of the attacks under a certain cache configuration.
- We systematically measure and analyze each cache parameter's influence on the attacks' success rate. Based on the measurement results, we find the private cache is the key to the success rates; the 8KB, 16-way private cache can achieve the optimal balance between the security and the cost. Although the shared cache's impacts are trivial, running neighbor processes can significantly increase the success rates of the attacks. The replacement policies and cache clusivity also have impacts on the attacks' performances: Random replacement leads to the highest success rates while the LFU/LSU leads to the lowest; the exclusive policy makes the attacks harder to succeed compared to the inclusive policy. We then use these findings to enhance both the cache side-channel attacks and defenses and strengthen future systems.

2 Background

In this section, we provide the knowledge about modern cache model and the GEM5 platform, and explain how the Bernstein's time-driven attack works.

2.1 CPU Cache Hierarchy

The CPU cache is in between of the CPU cores and the main memory. It stores copies of the frequently used data in main memory, aiming to reduce the average latency of data access from the main memory. Compared to the main memory, the cache is faster but smaller. Modern multi-core CPUs usually organize the cache as a hierarchy of multiple cache levels. Fig. 1 indicates a generic cache hierarchy model. Each CPU core is bound with a *private cache* and has this private cache's

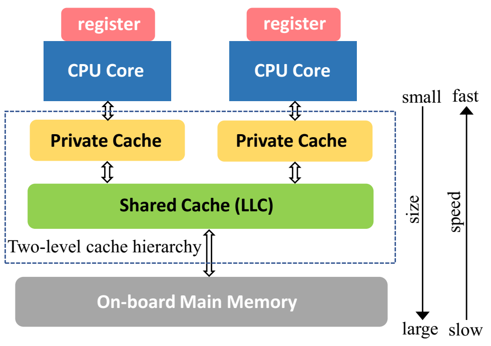


Figure 1: A generic cache model. It is a two-level hierarchy: each CPU core has its own private cache; all cores could access a shared cache through the private cache.

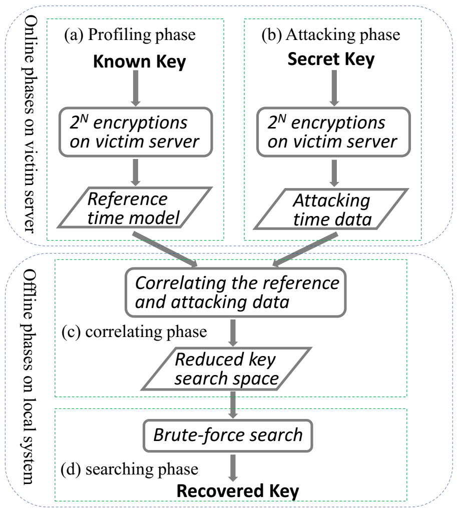


Figure 2: The workflow of Bernstein's time-driven attack. It consists of two online phases: (a) profiling phase and (b) attacking phase, and two offline phases: (c) correlating phase and (d) searching phase.

exclusive access permission. The whole CPU chip has one *shared cache* (a.k.a Last Level Cache (LLC)) that shared by all the CPU cores. The shared cache usually is larger but slower than the private cache. Commodity Intel CPUs follow this cache model and usually implement the private cache as two levels (L1 and L2) and split L1 cache into instruction and data caches. Private cache size typically is in KB scale while shared cache size is in MB scale.

2.2 Bernstein's Cache Timing Attack

The Bernstein's cache timing attack on AES [4] is one of the most classical time-driven cache attacks. It cracks the AES keys by measuring and analyzing the encryption time data.

Fig. 2 shows the workflow of Bernstein's attack. On the victim server, the attacker performs two online phases. During the *profiling phase* (Fig. 2 (a)), she or he encrypts millions of different plaintexts with a **known key**, then collects and profiles each execution time to build the reference time model. During the *attacking phase* (Fig. 2 (b)), she or he collects attacking time data by measuring the execution time of millions of encryptions with an **unknown key**. Then two offline phases postprocess the attacking data to guess the unknown key. Specifically, during the correlating phase(Fig. 2 (c)), the attacker mathematically correlates the time model and attacking data to shrink the key search space by generating value candidates for each key byte. Finally, in the searching phase(Fig. 2 (d)), the attacker brute-force searches the reduce key space to recover the secret key.

Only the two online phases need to be performed on the victim server. Although the Bernstein's attack is computationally expensive (typically needs 2^{22}-2^{23} encryptions for profiling and attacking phases respectively [4]), it does not require attacker's intervention during the attack. Thus, conducting this attack does not need much computer architecture knowledge and the victim system's access privilege.

2.3 GEM5 Platform

GEM5 is a modular platform for computer-system architecture research [6]. GEM5 is the combination of two influential projects: M5 [7] and Gems [24]. The former is renowned for CPU simulation while the latter is for memory system simulation. Therefore, GEM5 can cycle-accurately emulate the full X86 system encompassing system-level architecture and processor microarchitecture. It provides the interfaces to manipulate the configuration of almost all system components, including processing units, cache, and main memory. In the full-system mode, GEM5 runs a real operating system on it and allows users to interact with the OS. Hence, users can run any applications on GEM5 as running on real-world hardware. The downside of GEM5 is that its execution is 1000X slower than real hardware.

3 Measurement Design

We measure the performances of Bernstein's attacks running on different GEM5 instances. We configure these instances' caches using seven cache parameters, including *Private Cache Size*, *Private Cache Associativity*, *Shared Cache Size*, *Shared Cache Associativity*, *Cacheline Size*, *Replacement Policy*, and *Cache Clusivity*. We elaborate all the cache parameters in Section 3.1. GEM5 instances play the role of victim servers; hence we only need to run two online phases (Fig. 2 (a) and (b)) on it. Under each cache configuration, in accordance with the references [2, 4, 10], we conduct 2^{22} encryptions for profiling and attacking phases respectively. To quantify the performances of the attacks, we define the success rate of the time-driven attack in Section 3.2.

3.1 Cache Parameters for Configuration

GEM5 platform gives us the full privilege to configure the cache. Without loss of generality, we configure the cache in our measurements to a two-level hierarchy as shown in Fig. 1. Under such implementation, there are seven controllable cache parameters listed below:

1. **Private Cache Size (PCS)** The total amount of private cache space. It generally falls in the range from 2KB to 32KB. Intuitively, smaller PCS makes the attacks easier.
2. **Private Cache Associativity (PCA)** The number of cache blocks in a private cache set. Its range typically is between 2-way and 32-way. Intuitively, larger PCA results in lower success rates of the attacks.
3. **Shared Cache Size (SCS)** The total amount of shared cache space. It usually is a value in between 2MB and 32MB. Intuitively, the smaller SCS the easier attacks.
4. **Shared Cache Associativity (SCA)** The number of cache blocks in a shared cache set. It commonly is consistent with the PCA and intuitively has similar impacts.
5. **Cacheline Size (CLS)** The size of the basic unit of cache storage. The common CLSs are 32Bytes, 64Bytes, and 128Bytes. Intuitively, larger CLS leads to lower success rates of the attacks.
6. **Replacement Policy (RP)** The algorithms to manage the contents stored in the cache. The RP's impacts are highly-depend on the application's memory locality.
7. **Cache Clusivity (CC)** Clusivity describes whether the data in one level of cache present also in other levels. It is either inclusive or exclusive. Intuitively, the exclusive policy makes the attacks easier.

We reconfigure the cache of GEM5 platform through changing the above parameters. In the subsequent sections, we elaborate these cache parameters and their possible theoretical impacts on the time-driven attacks.

3.1.1 Private and Shared Cache Size

Both private and shared cache copy a portion of the main memory that possibly contains the next data that CPU will use. Apparently, larger PCS and SCS provide broader coverage and lower cache-miss rate; However, taking into account the production cost factor, commodity systems usually limit their PCS in KB scale and SCS in MB scale.

Theoretical expectation of PCS's impacts. We attack the AES implemented in OpenSSL library. OpenSSL precomputes the results of each AES step and stores them as a 4KB lookup table [23]. Given the time-driven attacks rely on the time differences of overall cache hits and misses, theoretically, a PCS less than 4KB renders higher miss rates and consequently causes higher success rates of the attacks. After 4KB, the attacks are impossible since the private cache can hold the entire lookup table.

Theoretical expectation of SCS's impacts. We discuss the impacts of SCS in two different scenarios. Theoretically, given the SCS is in MB scale that is much larger than the 4KB table, if there are no other processes running on the neighbor CPU cores, the SCS impacts to the attacks' success rates could be negligible. On the other hand, if there are random neighbor processes running, the SCS's impacts can be non-trivial. In the next section, we will experimentally examine whether the facts align with these theoretical expectations.

3.1.2 Private and Shared Cache Associativity and Cacheline Size

The basic data copying unit between caches and main memory is referred to as *cacheline*. A cacheline contains multiple bytes. The commodity CPUs usually divide the caches into multiple *sets* that each of them contains N cachelines (referred to as *N-way associativity*). Each cacheline of the main memory must go to a particular cache set but can choose one of the N positions in the set to reside. Either PCA or SCA (i.e., the N) is a trade-off between miss-rate and the searching cost.

Theoretical expectation of CLS, PCA, and SCA's impacts. The larger CLS leverages more spatial locality hence has lower miss-rates. The larger PCA leads to lower miss-rates as well. Lower cache miss-rates result in lower success rates of the attacks. Similar to SCS, if there are no running neighbor processes, SCA's impacts will be trivial. Otherwise, SCA's impacts are analogous to PCA's.

3.1.3 Replacement Policies

When a cache-miss occurs and the cache is full, the cache should make room for the new entry. Replacement policies are sophisticated algorithms to choose the existing cache entries to evict. Since the mission of the cache is storing the data that CPU is most likely to use next, the RPs speculate about which existing entries have the least chances to be used in the future. Given this heuristic is difficult, there is no universal optimal choice among available RPs.

In our measurement design, we examine four most common RPs: (i) *Least recently used (LRU)*. It keeps tracking what cache entries are used and discards the least recently used entry first. (ii) *Least-frequently used (LFU)*. It counts how many times each cache entry is used and discards the least often used entry first. (iii) *First in first out (FIFO)*. Cache with the FIFO works in the same way as FIFO queue. It always discards the entry loaded first. (iv) *Random replacement (RANDOM)*. It randomly chooses the candidate entries and evicts them when necessary. It doesn't require tracking any cache access history.

Theoretical expectation of RP's impacts. The relations of the RPs and the miss-rates highly depend on the memory access patterns of the crypto algorithms. Therefore, without the measurement data, it is difficult to predict the impacts of RPs on the time-driven attacks' success rates. We will use the measurement results in the next section to illustrate it.

3.1.4 Cache Clusivity

Cache Clusivity describes the data consistency policy between the private and shared cache. We consider two CCs: *inclusive* and *exclusive*. With the inclusive policy, all entries in the private cache will also present in the shared cache. Evicting a shared cache entry results in the eviction of the corresponding private cache entry. Whereas with the exclusive policy, the shared cache contains only the entries that do not present in the private cache. A hit entry in shared cache will be moved to private cache while an evicted entries from the private cache will be stored in the shared cache.

Theoretical expectation of CC's impacts. The inclusive policy implies less cache-miss penalties but smaller unique memory capacity. The exclusive policy is the opposite. Given the AES lookup table is small, cache-miss penalties will dominate the influences on the attacks' success rates. Hence we expect the inclusive policy leads to lower success rates compared to the exclusive policy.

3.2 Metric Definition: the Success Rate of an Attack

Conventionally, the metric for cache side-channel attack performance is a dualism: either success or failure. For any attack, no matter the difficulty, if it can crack the unknown key using reasonable computational resources and time, we label it with success; otherwise, we label it as a failure.

However, this binary metric does not have sufficient granularity to support performance comparison of the attacks labeled with success. To compare two successful attacks or evaluate a single successful attack running under different hardware environments, we need to measure the attacking difficulty i.e., the amount of demanded computational resources and time. Accordingly, we leverage the classical cryptanalysis concept [5] and propose the ***equivalent key length (EKL)*** to indicate the *success rate* of the time-driven cache side-channel attack. The *EKL* is formally defined as the following:

Definition 1. *The equivalent key length (EKL) is a normalized metric to represent the key search space. The EKL value* $\in [0,1]$ *is computed as in Equation 1, where n is the length of the target key,* v_k *denotes the number of candidate values for the k-th key byte after the correlating phase and* $k \in [0, n-1]$.

$$EKL = 1 - \frac{\sum_{k=0}^{n} \log_2 v_k}{8n} \tag{1}$$

In our measurement, we use 16-byte keys, i.e. n is 16. *EKL* being 0 represents the original search space, i.e. the correlating phase (Fig. 2 (c)) fails to reduce the number of candidate values of any key byte. On the other extreme, *EKL* being 1 represents the single element search space, i.e. the correlating phase fully reveals the unknown key.

We indicate the success rate of time-driven attacks using the ratio of *EKL* to the number of encryptions in attacking phase. Apparently, with the same amount of attacking encryptions, larger *EKL* implies a higher success rate. Increasing the number of encryptions could boost the *EKL*, however, will demand more measurement cost. One does not need to make the *EKL* achieve 1 at the correlating phase. We only need to reduce the search space to a reasonable size before entering the brute-force search phase (Fig. 2 (d)). Practically, an *EKL* at 0.8 achieves a good balance of the measurement cost and the brute-force search cost.

Table 1: Measurement Environment

Host System	
CPU	Intel(R) Xeon(R) E5-2620
main memory	192 GB RDIMM
GEM5 Platform	
CPU core #	2 cores (3 GHz)
main memory	4 GB
CPU cache	two-level configurable
operating system	Ubuntu 16.04.1 LTS
OpenSSL version	1.0.2 LTS

4 Measurement Results

We perform our measurements on a 10-node cluster. Each node is equipped with a 24-core Intel(R) Xeon(R) E5-2620 2.4GHz CPU and a 192GB RDIMM main memory. We launch multiple GEM5 instances. Each instance emulates the system with two 3GHz CPU cores, 4GB main memory, and a two-level cache. Different instances have different cache configuration; each configuration is one combination of seven cache parameters' candidate values described in Section 3.1. In each GEM5 instance, we run a Bernstein's time-driven attack to attack the AES implemented in OpenSSL 1.0.2. This OpenSSL version uses the lookup table and is vulnerable to cache timing side-channel attacks. The measurement environment is summarized in Table 1.

We measure the attack's success rates under every instance and show them in Fig. 3, then analyze the impacts of each cache parameter respectively. In a nutshell, the private cache has a significant effect on the attacks' success rates; larger private cache size and associativity could make the attacks more difficult. The shared cache parameters have minor impacts on the attacks without regard to the neighbor processes. The replacement policies can significantly affect the success rate of attacks while the cacheline size's effect is insignificant. The clusivity also influences the success rates; using the exclusive policy makes the attack harder than using the inclusive policy. We elaborate on the impacts of each cache parameter in the following sections.

4.1 Private Caches' Impacts

PCS Fig. 3a shows how the Private Cache Size impacts the success rates of the attacks. We can observe that the overall trend of the impact is that the larger PCS leads to a lower success rate. Moreover, we can see a cliff-like drop in the success rate when the PCS increases from 4KB to 8KB.

The observed fact, by and large, matches the theoretical expectation stated in Section 3.1.1. The larger PCS results in fewer cache-misses (i.e., fewer time differences) and makes the time-driven attacks more difficult. The cliff drop between 4KB and 8KB is caused by the AES lookup table size (4KB). When the private cache is larger than 4KB, since there are no other processes co-located on the same core, the entire lookup table theoretically can reside in the private cache. It means the cache-misses will not happen; hence, the time-driven attacks will not succeed. However, disagreed with the expectation, the measurement result shows that, although it is much harder, the attacks still have the chance to succeed with a PCS larger than 4KB. It implies that the AES computation itself and the system operations can kick some lookup table entries out of the private cache.

PCA Fig. 3b indicates the attacks' success rates influenced by the Private Cache Associativity. We observe that the success rates and the PCA are negatively correlated: larger PCA leads to lower success rates. We also find that there is a sharp success-rate decrease from 8-way to 16-way. The 16-way and 32-way associativities make the attacks have nearly the same success rates.

For the most part, the impact of PCA matches the expectation discussed in Section 3.1.2. A larger PCA results in less cache misses hence makes the attacks harder to succeed. The experimental results imply that if a cache set has 16 entries or more, the loaded entry mostly can find an appropriate place in the set without flushing the data needed by next cache reads. Accordingly, from 8-way to 16-way, the cache-misses become very occasional. It causes that the success rates fall quickly from 8-way to 16-way, and 16-way and 32-way make similar success rates.

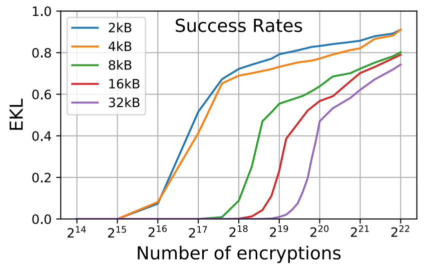
Above findings suggest that the private cache parameters have significant influences on the success rates of the time-driven attacks. The influences mostly comply with the theoretical expectations with a few unexpected singular points.

4.2 Shared Caches' Impacts

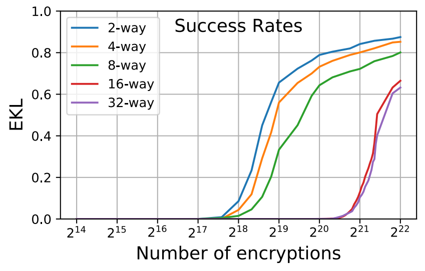
SCS We use the *stress* tool to generate random memory-intensive workloads and run these workloads as the neighbor processes. Fig. 3c and 3d show the attacks' success rates impacted by Shared Cache Size without and with the neighbor processes (NP) respectively. As anticipated, without NP, the SCS has negligible impacts on the success rates. Differing from the expectation, although running the NP can make the attacks easier, changing SCS still has no significant impact on the success rates. The reason is likely to be that the MB-level shared cache is huge compared to 4KB AES lookup table; so the table entries always have similar chances to be found in shared cache no matter it is 4MB or 32MB.

SCA Fig. 3e and 3f are the success rates of attacks impacted by various Shared Cache Associativity without and with the NP respectively. SCA without the NP matches the expectation: it barely affects the success rates. On the contrary, similar to SCS, despite the NP can increase the attacks' success rates, the SCA's impacts are still trivial.

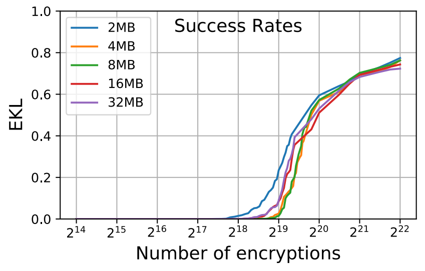
Our findings in this suction suggest that shared cache parameters have no significant impact on the attacks' success rates with no regards of the NP. As distinct from the theoretical expectation, although running a memory-intensive NP can make the attacks easier to succeed, it cannot boost either SCS or SCA's impacts on the attacks' success rates.



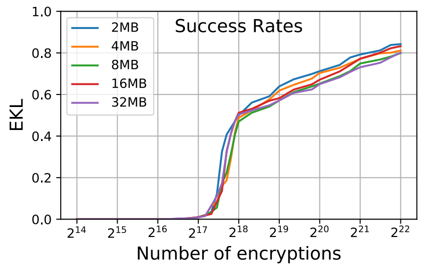
(a) Private Cache Size



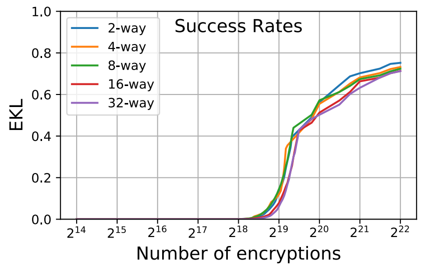
(b) Private Cache Associativity



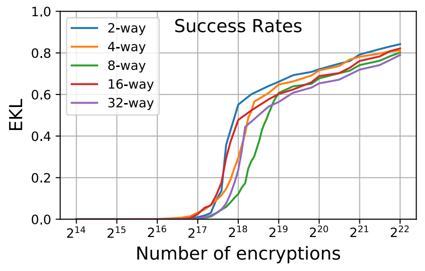
(c) Shared Cache Sizes (w/o NP)



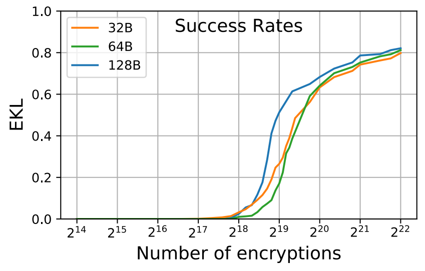
(d) Shared Cache Sizes (w/ NP)



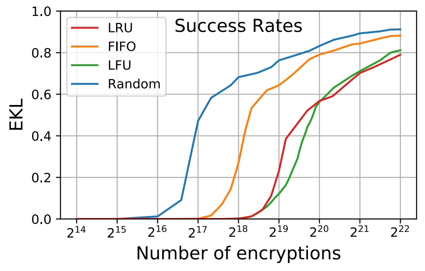
(e) Shared Cache Associativity (w/o NP)



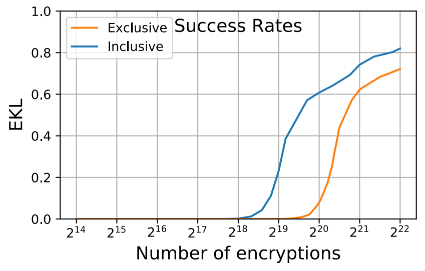
(f) Shared Cache Associativity (w/ NP)



(g) Cacheline Sizes



(h) Replacement Policies



(i) Cache Clusivity

Figure 3: The success rates of time-driven attacks impacted by different cache parameters. NP stands for the neighbor processes running on the neighbor CPU core. The x-axis represents the numbers of encryptions conducted during the attacking phase (Fig. 2(b)). The y-axis indicates the equivalent key lengths (Section 3.2).

4.3 CLS, RPs, and CCs's Impacts

CLS We measure the attacks' success rates affected by different CacheLine Sizes. Disagreeing with the expectation in Section 3.1.2, Fig. 3g indicates that the impacts of CLS are subtle and random. There are two possible explanations. One reason is that the AES computation has good spatial locality that many next cache-reads are within 32B. The other is that the AES has bad locality that many next reads are out of the 128B range. Later we will indicate that the former explanation matches the fact. So changing the CLS will not significantly change the cache-miss rates.

RPs We also measure the attacks' success rates under a variety of Replacement Policies described in Section 3.1.3. Fig. 3h indicates the RPs have significant impacts on the success rates. The attacks under RANDOM have the highest success rates while under FIFO have the runner-up ones. The LFU and LRU lead to similar success rates that are lower than the other two RPs.

The measurement results imply that AES computation has good temporal and spatial localities. It is consistent with the findings regarding CLS. So the LFU and LRU result in the least miss-rates, i.e. lowest success rates of the attacks. Since the RANDOM does not leverage any localities, many cache misses can occur, makes the attacks easy to succeed. The FIFO leverages the localities to some extent, makes its success rates lie between the RANDOM and the LFU/LRU.

CCs We finally measure the Clusivity's impacts. In contrast to the expectation described in Section 3.1.4, Fig. 3i shows the exclusive policy makes the attacks more difficult to succeed. The reason is that private cache's cache-misses dominate the AES computation time. Since the exclusive policy stores all private cache evicted entries into the shared cache, the private cache miss rates are low. Conversely, the inclusive policy removes the private cache entries whenever their copies in the shared cache are evicted, hence increases the private cache miss rates and makes the attacks easier.

5 Discussion

1) Takeaways *Private cache configuration is the key to the success rates of the time-driven cache attacks.* Although larger PCS and PCA cannot completely prevent the time-driven attacks, they can make the attacks much harder to succeed.

Shared cache configuration is trivial to the attacks' success rates, but the neighbor processes can have significant impacts on the success rates. A memory-intensive neighbor process can make the attacks one order of magnitude easier.

Replacement policies and cache clusivity also can influence the attacks' success rates. In this Bernstein's AES attack case, the random replacement leads to the easiest attack; the exclusive policy makes the attack easier compared to the inclusive policy.

2) Suggestions for the attackers Increasing the eviction rate of the AES lookup table entries from the private cache can make the time-driven attacks easier to succeed. A feasible way is binding a noise process with the same CPU core running the AES encryptions. It leads to a higher possibility of kicking the table entries out of the private cache.

3) Suggestions for the defenders A luxury private cache definitely can reduce the vulnerability to time-driven cache attacks. However, in order to achieve the optimal cost-efficiency balance, it is better to set the private cache parameters at some inflection points, for example, 8KB PCS and 32-way PCA for this AES attack case.

If AES lookup table size can fit into private cache, using lock-into-cache instruction to ensure the entire table in the private cache can sharply reduce the attacks' success rates. Besides, assigning a CPU core exclusive and reserving a shared cache space for the AES encryptions can also make the cache less vulnerable to the time-driven attacks.

It is not necessary to keep the replacement policy and clusivity consistent between the private caches and shared cache. One can use Random replacement and exclusive policy for one private cache used by AES encryptions, then apply other replacement policies and clusivity to remaining private caches and shared cache, to balance the cache-attack resistances and the system performance efficiency.

6 Limitations

There are a few limitations in our measurement design from both the attacks and the systems aspects.

It is unclear whether our measurement design is compatible with other cache side-channel attacks. The targeted Bernstein's attack relies on the statistical patterns of the encryption time rather than any cache-behavior based analytic attack models. Its advantages include that it is portable between different systems with rare adaptations, and its performance is easy to be measured. However, some other time-driven attacks use the attack models based on particular cache effects, for example, the cache-collision effect [8]. The current measurement design may need case-by-case modifications for investigating these specific time-driven attacks. The current design also may be not compatible with other crypto algorithms, e.g. DES, RSA and with the access-driven attacks.

Our measurement design does not count the effects of some modern hardware technologies. The GEM5 platform accurately emulates the cache latency and behaviors, but it does not implement some advanced hardware techniques, e.g. the prefetcher. The prefetcher predicts the future cache accesses and loads the data to the cache before any possible cache-misses happen. It may significantly change the impacts of some cache parameters, including cacheline sizes and replacement policies, on the attacks' success rates. It is difficult to theoretically model the prefetcher's effects since it depends on the prediction accuracy for specific encryption algorithms.

7 Related Work

Time-driven attacks. Bernstein's attack [4] is one of the most classical practical implementations of time-driven attacks. It is elaborated in Section 2.2. Bonneau and Mironov [8] proposed a finer-grained time attack. They observed the time variations caused by cache-collisions happened during table lookups of encryption. This attack requires less computational cost but is more difficult to measure. Tiri et al. [27] proposed an analytic model for time-driven attacks that can estimate the strengths of the symmetric key cryptosystems. Brumley and Hakala [9] proposed a timing template attack. They combined vector quantization and the hidden Markov model to build a tool to automatically analyze the cache-timing data. Brumley and Tuveri [10] described the timing attack vulnerability in OpenSSL implementation and approved that the time-driven attack on a remote server is feasible. Wang et al. [28] proposed CacheD, a software examining technique that leverages symbolic execution to identify potential cache time differences at each program point.

Access-driven attacks. Yarom et al. [29] presented FLUSH + RELOAD attack. This attack targets LLC and can recover the keys even if the processes are not co-located in the same core. Liu et al. [20] implemented the PRIME + PROBE attack also targeting on the LLC. This attack is cross-core, cross-VM, and works on multiple versions of GnuPG without relying on weaknesses of OS or VM. Irazoqui et al. [16] introduced another variation of PRIME+PROBE attack. Their attack is a fine-grain cross-core LLC attack. It needs no deduplication and OS configuration changes hence is quite viable. Gruss et al. [15] proposed the access-driven template attack on shared inclusive LLC. It automatically profiles and exploits cache leakage of any programs without requiring prior knowledge of specific software and system information. Gras et al. [13] proposed TLBleed that attacks the page-table caches (TLB) to bypass the hardware cache side-channel protections.

Implementations' and systems' impacts. To our best knowledge, our work is the first systematic and experimental study of the cache configurations' impacts on the cache attacks' performances. The most relevant existing work we are aware of was done by Mantel et al [23]. Although the crypto algorithms are strong, careless implementations and misuses can make the ciphers vulnerable [25, 26]. Mantel et al. systematically studied how different AES implementations influence the vulnerability to cache side-channel attacks. They leveraged the CacheAudit static analyzer [12] to check the leakage bounds of different AES implementations. In [23], only one cache parameter (PCS) was considered. Moreover, its results are still counted as the theoretical expectations since CacheAudit is a program-analysis-based tool that calculates the leakage bounds without any actual executions.

8 Conclusion

In this paper, we systematically studied how cache configurations impact the success rates of time-driven cache attacks. We addressed the difficulty of conducting apples-to-apples experimental comparisons and proposed the methodology to measure the attacks' performances under different cache configurations. In our measurement design, we made the cache-attack performances comparable by extending the traditional success-fail binary metric to the quantifiable success rate metric. We leveraged the GEM5 platform to emulate the X86 system with the configurable cache. We configured the cache through seven cache parameters, including Private and Shared Caches' Size and Associativity, Cacheline Size, Replacement Policy, and Clusivity. From the measurement results, we found that the private caches' impacts on the attacks' success rates are significant while the shared caches' are trivial; the replacement policies and cache clusivity also have clear influences on the attacks' performances. We provided suggestions to the attackers and defenders and implications for the future system designs according to our measurement findings.

Our measurement work is focused on cache timing based side channels. It remains an interesting open question of whether one can similarly characterize other types of more complex side channels in a systematic fashion.

9 Future Research Directions

We can extend our measurement to answer some other interesting in-depth questions.

Are this measurement's approach, findings, and conclusions transferable to other cache side-channel attacks? To answer this question, we should extend our experiment to measure the performances of other side-channel attacks. However, as mentioned in Section 6, we might need to modify the current measurement design to fit other attacks. For example, with the purpose of measuring the performance of the access-driven attacks on GEM5, one needs to run malicious processes concurrently with the victim crypto process on the GEM5 platform.

Do the RISC-based embedded systems' cache configurations have the same impacts? In order to answer it, we plan to measure the attacks' performance on the Rocket emulator [3] whose cache also can be configured. Different from the GEM5 that implements the CISC ISA, the Rocket implements the RISC-V specification and can emulate the SoC embedded platforms. This extension will complete our measurement by covering all scale systems from embedded to server systems.

How accurate is this emulation-based measurement? Can we apply the conclusions to predict the cache timing attack vulnerability of unseen systems? Though the GEM5-based measurement results can largely reflect the real hardware's impacts, according to the limitation stated in Section 6, they might not exactly match the actual performances on modern commodity CPUs. In the future, we can run the attacks on real CPUs with corresponding cache configurations to examine the deviations, and then try to use the statistic approaches to model these deviations. Moreover, inspired by the works that model and predict the system performance variability [11, 21, 22], we can build the vulnerability prediction model based on the measurement and deviation analysis results. The possible linear and nonlinear statistic tools that we can leverage to build the model include linear Shepard and Delaunay triangulation. This model will be useful to predict the success rates of cache attacks on the unseen systems. The predictions can provide implications for future system designs to reduce the cache-attack vulnerability.

Acknowledgments

This work is sponsored in part by the NSF under the grant CNS-1565314/1838271. The authors thank the whole VarSys team, including Thomas Lux, Bo Li, Jon Bernard, Chandler Jearls, Li Xu, Tyler Chang, Prof. Ali Butt, Prof. Yili Hong, Prof. Layne Watson, Prof. Godmar Back, and Prof. Margaret Ellis, for their valuable comments and discussions. The authors acknowledge the help about GEM5 usage from Da Zhang and Prof. Xun Jian.

References

[1] M. Alian, A. H. Abulila, L. Jindal, D. Kim, and N. S. Kim. Ncap: Network-driven, Packet Context-aware Power Management for Client-Server Architecture. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 25–36. IEEE, 2017.

[2] H. Aly and M. ElGayyar. Attacking AES Using Bernstein's Attack on Modern Processors. In *Progress in Cryptology – AFRICACRYPT 2013*, pages 127–139. Springer, 2013.

[3] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz,

S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman. The Rocket Chip Generator. Number UCB/EECS-2016-17, Apr 2016.

[4] D. J. Bernstein. Cache-Timing Attacks on AES. 2005.

[5] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-round DES. In *Advances in Cryptology — CRYPTO' 92*, pages 487–496. Springer, 1993.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[7] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, (4):52–60, 2006.

[8] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 201–215. Springer, 2006.

[9] B. B. Brumley and R. M. Hakala. Cache-Timing Template Attacks. In *Advances in Cryptology – ASIACRYPT 2009*, pages 667–684. Springer, 2009.

[10] B. B. Brumley and N. Tuveri. Remote Timing Attacks Are Still Practical. In *Computer Security – ESORICS 2011*, pages 355–371. Springer, 2011.

[11] K. W. Cameron, A. Anwar, Y. Cheng, L. Xu, B. Li, U. Ananth, J. Bernard, C. Jearls, T. Lux, Y. Hong, L. T. Watson, and A. R. Butt. Moana: Modeling and Analyzing I/O Variability in Parallel System Experimental Design. *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[12] G. Doychev, D. Feld, B. Kopf, L. Mauborgne, and J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 431–446. USENIX, 2013.

[13] B. Gras, K. Razavi, H. Bos, and C. Giuffrida. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 955–972. USENIX Association, 2018.

[14] D. Gruss, C. Maurice, K. Wagner, and S. Mangard. Flush+Flush: A Fast and Stealthy Cache Attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer, 2016.

[15] D. Gruss, R. Spreitzer, and S. Mangard. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 897–912. USENIX Association, 2015.

[16] G. Irazoqui, T. Eisenbarth, and B. Sunar. S$A: A Shared Cache Attack that Works Across Cores and Defies VM Sandboxing–and its Application to AES. In *2015 IEEE Symposium on Security and Privacy*, pages 591–604. IEEE, 2015.

[17] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO '96*, pages 104–113. Springer, 1996.

[18] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990. USENIX Association, 2018.

[19] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee. Catalyst: Defeating Last-Level Cache Side Channel Attacks in Cloud Computing. In *2016 IEEE international symposium on high performance computer architecture (HPCA)*, pages 406–418. IEEE, 2016.

[20] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, 2015.

[21] T. C. Lux, L. T. Watson, T. H. Chang, J. Bernard, B. Li, X. Yu, L. Xu, G. Back, A. R. Butt, K. W. Cameron, D. Yao, and Y. Hong. Nonparametric Distribution Models for Predicting and Managing Computational Performance Variability. In *SoutheastCon 2018*, pages 1–7. IEEE, 2018.

[22] T. C. H. Lux, L. T. Watson, T. H. Chang, J. Bernard, B. Li, X. Yu, L. Xu, G. Back, A. R. Butt, K. W. Cameron, D. Yao, and Y. Hong. Novel Meshes for Multivariate Interpolation and Approximation. In *Proceedings of the ACMSE 2018 Conference*, ACMSE '18, pages 13:1–13:7. ACM, 2018.

[23] H. Mantel, A. Weber, and B. Köpf. A Systematic Study of Cache Side Channels Across AES Implementations. In *Engineering Secure Software and Systems*, pages 213–230. Springer, 2017.

[24] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[25] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, D. D. Yao, and M. Kantarcioglu. CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019.

[26] S. Rahaman and D. Yao. Program Analysis of Cryptographic Implementations for Security. In *2017 IEEE Cybersecurity Development (SecDev)*, pages 61–68. IEEE, 2017.

[27] K. Tiri, O. Acıiçmez, M. Neve, and F. Andersen. An Analytical Model for Time-Driven Cache Attacks. In *Fast Software Encryption*, pages 399–413. Springer, 2007.

[28] S. Wang, P. Wang, X. Liu, D. Zhang, and D. Wu. CacheD: Identifying Cache-based Timing Channels in Production Software. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 235–252, 2017.

[29] Y. Yarom and K. Falkner. FLUSH+ RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security Symposium*, pages 719–732, 2014.

[30] D. Zhang, V. Sridharan, and X. Jian. Exploring and Optimizing Chipkill-Correct for Persistent Memory Based on High-Density NVRAMs. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 710–723, 2018.