How secure is your cache against side-channel attacks?

Zecheng He
Princeton University
Princeton, NJ
zechengh@princeton.edu

Ruby B. Lee
Princeton University
Princeton, NJ
rblee@princeton.edu

ABSTRACT

Security-critical data can leak through very unexpected side channels, making side-channel attacks very dangerous threats to information security. Of these, cache-based side-channel attacks are some of the most problematic. This is because caches are essential for the performance of modern computers, but an intrinsic property of all caches – the different access times for cache hits and misses – is the property exploited to leak information in time-based cache side-channel attacks. Recently, different secure cache architectures have been proposed to defend against these attacks. However, we do not have a reliable method for evaluating a cache's resilience against different classes of cache side-channel attacks, which is the goal of this paper.

We first propose a novel probabilistic information flow graph (PIFG) to model the interaction between the victim program, the attacker program and the cache architecture. From this model, we derive a new metric, the Probability of Attack Success (PAS), which gives a quantitative measure for evaluating a cache's resilience against a given class of cache side-channel attacks. We show the generality of our model and metric by applying them to evaluate nine different cache architectures against all four classes of cache side-channel attacks. Our new methodology, model and metric can help verify the security provided by different proposed secure cache architectures, and compare them in terms of their resilience to cache side-channel attacks, without the need for simulation or taping out a chip.

CCS CONCEPTS

• **Security and privacy** → **Side-channel analysis and countermeasures**; • **General and reference** → *Evaluation*; • **Computer systems organization** → *Processors and memory architectures*;

KEYWORDS

Cache, side-channel attack, security modeling, quantification

ACM Reference Format:
Zecheng He and Ruby B. Lee. 2017. How secure is your cache against side-channel attacks?. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 13 pages.
https://doi.org/10.1145/3123939.3124546

1 INTRODUCTION

Confidentiality, integrity and availability are the key components of information security. Among them, loss of confidentiality cannot be reversed once the information has leaked. Encryption is very frequently used to protect data confidentiality. Various encryption algorithms have been proposed, as have attacks on these algorithms. Some of these attacks focus on intrinsic algorithm vulnerabilities and try to break the algorithms themselves. Other attacks target implementations of the algorithms, even though the algorithms are shown to be secure. Side channel attacks fall in the latter class, and they target implementation characteristics, such as execution time [8], power consumption [21], resource sharing [9], sounds [1] and radiation [10] of real implementations of crypto algorithms.

Side channel attacks are serious threats to information security for many reasons. First, side-channel attacks are hard to detect. Most side-channel attacks do not require special privileges or equipment, and their behavior may not be overtly harmful. For example, a simple program running in user mode measuring execution time, or a smartphone near a laptop with its microphone on, can both be side-channel attacks. Second, side channels are not easy to eliminate without affecting performance. Removing potential side channels from physical or software implementations of crypto algorithms often significantly degrades efficiency, e.g., disabling a cache or disabling sharing of bus wires. Third, side-channel attacks can break various systems. This is because side-channel attacks target the physical features of implementations of algorithms, rather than a specific algorithm. Therefore, different crypto algorithms can be the targets of one type of side-channel attack, as long as they have some similarity in their hardware or software implementations. Fourth, side-channel attacks destroy the entire confidentiality of data or programs, once they succeed, because they recover cryptographic keys, instead of decrypting specific data. No matter what plaintext is encrypted, as long as the secret keys are leaked, the encrypted data can be easily recovered by attackers because encryption/decryption algorithms are publicly known.

Cache based side-channel attacks exploit the difference in the access times of cache hits and misses. This makes cache side-channel attacks hard to eliminate, because this time difference is an inherent feature of all caches. Also, disabling the cache is not a practical solution to these attacks, because caches are the most important performance component in modern computers, and disabling the cache causes unacceptable performance degradation.

1.1 Past Work

These cache attacks are realizable in the real world, for example [3], [8], [2], [25] successfully attack the L1 cache, and recovered the secret keys of AES. [20] shows cache side-channel attacks are practical on the last level cache (LLC) as well. [33] shows that L2 cache

MICRO-50, October 14–18, 2017, Cambridge, MA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3124546

side-channel attacks in virtualized environments are practical. [37] successfully launched cache side-channel attacks in cloud settings.

Many approaches have been proposed to defeat these attacks, and they can be categorized into two classes: software based approaches [4], [11] and hardware based approaches. Software approaches often mitigate only specific applications. For example, [23] discusses compact S-box tables and frequently randomized tables for AES encryption. Special instructions are proposed for AES in [22] and [28]. System level mitigation [12] may cause impact on performance. Uncachable tables and inaccessible high-resolution timers are approaches to defend against side-channel attacks [29], [30]. However, uncacheable tables increase the total execution time of encryption and inaccessibility of timers cause other benign applications to not work properly. [34] and [16] claim prefetching (and locking) can be used as a mitigation technique, without significantly influencing performance, however, only a specific type of attack can be prevented. [16] proposes a defense of LLC (Last Level Cache) side-channel attacks based on CAT (Cache Allocation Technology). This approach not only prefetches secure critical data, but also pins them in the LLC, thus all secure critical accesses are hits. However, pinning the data in the cache causes cache fragmentation and needs support by special hardware.

Hardware based approaches provide a different way of defending against cache side-channel attacks. Many new cache architectures have been proposed. In general, there are two categories of defenses: partitioning based approaches and randomization based approaches. The first category includes the Static Partitioned (SP) cache [24], the Partition Locked (PL) cache [31] and the Non-monopolizable (Nomo) cache [7]. Randomization based cache architectures include Random Permutation (RP) [31] cache, Newcache [32], [19], Random Fill (RF) [18] cache and Random Eviction (RE) cache [5]. We will describe these secure cache proposals in greater detail in Section 2, and will evaluate their resilience to cache side-channel attacks in Section 4.

Some work has been proposed on evaluating cache security. [27], [15], [14], [35] use mutual information to evaluate cache security. [5] introduces the side-channel vulnerability factor (SVF) and pattern correlation to evaluate caches. [36] suggests the Cache Side-channel Vulnerability metric (CSV) as an improvement to SVF for the access-based attacks considered in [5]. [27], [26] and [6] used success probability to calculate an attacker's chances of success.

However, very limited work has been proposed on systematically modeling and comparing between different caches against different kinds of attacks. In this paper, we will answer two questions: (1) How can we systematically model and evaluate a cache's resilience to side-channel attacks? (2) Among all proposed secure cache designs, which one is more resilient to a class (or classes) of cache side-channel attacks?

Our contributions are:

- We propose the Probabilistic Information Flow Graph (PIFG), a new model based on an information flow graph (IFG) and conditional probabilities, and a new metric, Probability of Attacker's Success (PAS) to evaluate and quantify a cache architecture's resilience to cache side-channel attacks. We show how PIFG models the relevant features of attackers, victims and cache architectures. We also show the impact of an attacker preparing for a cache attack (pre-PAS).
- We apply PIFG and PAS on nine cache designs and four practical attacks, which cover all the known time-based cache side-channel attacks and hardware secure cache defenses.

2 BACKGROUND

2.1 Cache Side Channel Attacks

Table 1 shows a classification of all cache side-channel attacks, using timing differences (due to cache hits or misses) as the channel medium to leak secret meta-data.

The intrinsic characteristic of most cache attacks is the timing difference between cache hits and misses. An ideal attacker is able to observe whether each of the victim's memory accesses is a cache hit or cache miss. He is also able to infer the memory address or addresses used by the victim from his observations of the cache shared with the victim. This memory address (or addresses) is the metadata leaked to the attacker through such cache timing side channel attacks, from which he can derive, for example, the secret key of encryption algorithms.

However, in general, attackers do not have the ability to observe each of the victim's cache accesses. Therefore, indirect observations are used. In some of these attacks, attackers measure the access time of each of his own individual memory accesses (column 2 of Table 1), after some interferences with the victim. In other attacks, attackers observe the total execution time of the victim's security-critical operation (column 1 of Table 1), instead of the access time of each memory access. In Table 1, we use the historical names, "access-based" versus "timing-based" cache side-channel attacks to distinguish these two types of attacks, although both are based on measuring time differences, but at different scales of a single memory access or the time for an entire security-critical operation (e.g., encryption of a whole block of memory), respectively.

Cache side-channel attacks can also be classified into "miss based" attacks versus "hit based" attacks. In the former, attackers are interested in observing whether there is a longer average time (access time or operation execution time, over many trials) due to cache misses. The cache misses are due to either the victim (column 1) or the attacker process (column 2).In the latter, denoted hit-based attacks, attackers are interested in observing whether there is a shorter average time due to cache hits.

Since hit/miss based attacks and timing/access based attacks are orthogonal, we have four categories of cache side-channel attacks, which cover the space of all known cache side-channel attacks. Our proposed model covers all these 4 categories, and thus all cache side-channel attacks based on the timing channel. We give a representative example of each category in Table 1, and describe each of these below. In the names of the representative attacks, the attacker performs the 2 actions, e.g., "evict" and "time", with the "-and-" indicating the attacker waits in-between these actions for the victim to perform the security-critical operation (e.g., crypto) between these two actions, except for the Collision attack, where the attacker does not perform any cache actions.

Type 1, e.g., Evict-and-time Attack: The attacker wants to observe whether the victim has a longer average execution time due

Table 1: Types of cache side-channel attacks. This table covers all known cache side-channel attacks due to time differences.

	Timing Based Attacks	Access Based Attacks
Miss Based Attacks	**Type 1:** Observe if victim uses the attacker-evicted cache line(s), causing victim's longer execution time for an entire security-critical operation. ***E.g., Evict-and-time attack***	**Type 2:** Observe if victim evicts the attacker's cache line(s), causing the attacker to later miss on these cache line(s) resulting in longer memory access time. ***E.g. Prime-and-probe attack***
Hit Based Attacks	**Type 3:** Observe if victim reuses memory lines fetched by his previous memory accesses, causing victim's shorter execution time for an entire security-critical operation. ***E.g. Cache Collision attack***	**Type 4:** Observe if attacker uses the victim-fetched cache line(s), which causes the attacker's shorter memory access time. ***E.g. Flush-and-reload attack***

to the victim's cache miss(es), over a large number of trials. In this attack, the attacker first evicts one (or some) cache set (sets) which contain the victim's security-critical data. If the victim uses the evicted data, a cache miss occurs. This cache miss increases the victim's execution time for the security-critical operation (e.g., a block encryption in AES), which can be observed by the attacker.

Type 2, e.g., Prime-and-probe Attack: The attacker wants to observe whether he gets a longer access time due to a cache miss for his own memory access. In this attack, the attacker first "primes" some cache lines by loading his own memory lines into the cache. Then, the victim executes the secure operation. If the victim's security-critical data maps to a primed cache line, the attacker's cache line is evicted. Then, when the attacker "probes" the same memory line(s) by loading it (them) again, he can measure his own access time to see whether he gets a cache miss or not. If the victim did not fetch his own data line into a primed cache slot, the attacker should get a cache hit.

Type 3, e.g., Cache Collision Attack: The attacker wants to observe whether the victim has a shorter execution time due to cache hits of the victim's own memory accesses. This attack is very special because the attacker does not need to interfere with the victim. When the victim accesses the same memory line M again, the second memory access is a hit because the first memory access has fetched M into the cache. This hit causes the victim's execution time to be shorter. The hits because of previous memory accesses are called "cache collisions".

Type 4, e.g., Flush-and-reload Attack: The attacker wants to observe if he gets a cache hit for his own memory access. In this attack, the attacker shares a library or data with the victim. The attacker first flushes the shared memory line(s) out of the cache, then waits for the victim to execute. If the victim uses the shared library or data, these shared memory lines will be fetched into the cache. After the victim finishes his security-critical operations, the attacker reloads the shared memory lines. A hit in the attacker's reloads indicates that the corresponding memory line has been used by the victim.

2.2 Secure Cache Architectures

Secure cache architectures have been proposed to defend against cache side-channel attacks. In general, they can be classified into two categories: partitioning based architectures and randomization based architectures. We briefly introduce these secure cache designs below.

2.2.1 Partitioning based architectures. Intuitively, partitioning based secure cache architectures separate the victim's cache partition and the attacker's cache partition. These architectures significantly reduce interference between the attacker's and the victim's memory accesses, however, they often degrade the cache's performance.

Static Partitioning (SP) Cache: This is the basic design of partitioning based caches. A static partitioning cache statically separates the cache for the victim and the attacker. No sharing of the cache is allowed in this architecture. This cache architecture prevents all interferences of victim's and attacker's memory accesses. However, the security protection is at the cost of degrading cache performance. It is only deployed in extremely security-sensitive applications.

Partition Locked (PL) Cache: Partition locked cache [31] leverages cache line based partitioning, which is much finer in granularity (better) than SP cache. Every cache line in the PL cache can have its Protection bit set or cleared. A protected cache line cannot be replaced by non-protected lines, e.g. by the attacker's data. On an attacker's cache miss to evict a protected cache line, the attacker's data is directly transferred from memory to the processor, without being brought into the cache. The line based granularity makes cache sharing still possible. The correct way to use PL cache is prefetching (and lock) all security-critical data into the cache before starting security-critical operations.

Non-monopolizable (Nomo) Cache: Non-monopolizable (Nomo) Cache [7] uses way-based partitioning, rather than set-based (for SP) or line-based (for PL) partitioning. Each security-sensitive process is assigned some reserved ways in each cache set. All the remaining cache ways can be shared by all processes. This modification prohibits the attacker occupying a whole cache set. However, if the victim's data exceed the reserved ways, the victim has to interfere with the attacker. In this situation, it is possible that the victim still leaks some information through cache sharing.

2.2.2 Randomization based architectures. Randomization based cache architectures mess up the information obtained by the attacker. We briefly discuss some randomization based cache architectures below. In general, randomization based architectures are more efficient, performance wise, than partitioning based approaches.

Conventional Fully Associative (FA) Cache with Random replacement policy: A fully associative cache with a random replacement policy is the most basic version of a randomization based cache. Any memory line can be mapped to any cache line, and a random cache line is evicted on a cache miss. However, FA caches are slow and power-hungry.

Random Permutation (RP) Cache: A Random permutation cache [31] uses dynamic permutation tables to map memory addresses to cache sets. In a conventional set-associative cache, the index bits in the memory addresses determine the cache set index number. In RP cache, each (secure) process has a distinct permutation table. Index bits are first translated to the real cache set index number using the table. If the victim's memory access M is mapped to a cache set S that belongs to another process (possibly the attacker), a random cache set S' will be evicted and the index mappings of S and S' are swapped.

Newcache: Newcache [32] [19] randomizes memory-to-cache mappings. This randomization is achieved by using an ephemeral logical cache between memory addresses and real physical cache lines. The mapping between memory addresses and the logical cache is like a direct mapped cache. The mapping between the logical cache and the physical cache is fully associative. Newcache is a power-efficient version of a fully-associative cache with random replacement because fewer tag bits need to be compared associatively. This reduces the width of the CAM (Content Addressable Memory) used to inversely map each physical cache line to a memory line [19].

Random Fill (RF) Cache: Random fill cache [18] modifies the fetch policy on a cache miss. On a cache miss, instead of bringing the accessed memory line M into the cache, RF cache randomly selects a memory line M' within a neighborhood window of M (from memory line $M - W_a$ to memory line $M + W_b$) with equal probability. Because this cache design only modifies the fetch policy, it can be combined with most of the other secure cache architectures.

Random Eviction (RE) Cache: Rather than randomizing the fetch policy or the memory-to-cache mapping, the Random Eviction (RE) cache randomly evicts cache lines out of the cache. A random cache line is evicted after a predetermined number of memory accesses. For example, 20% random eviction RE cache means the cache randomly evicts a cache line every 5 memory accesses.

3 PROBABILISTIC INFORMATION FLOW GRAPH

In this section, we define our proposed probabilistic information flow graph (PIFG) to model the cache architectures and cache side-channel attacks. However, since a PIFG can model more attacks than just cache side-channel attacks, we define PIFG more generally in this section. A probabilistic information flow graph is a directed acyclic graph (DAG) with edge flow probabilities on its edges. The information flow graph is used to model the attack and any interference with the victim, i.e. each type of the attack can have a unique information flow graph. The edge flow probabilities are used to model the different cache architectures. We define the terminology used in the probabilistic information flow graph below.

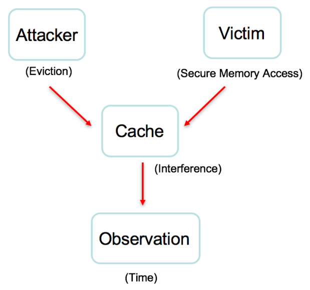


Figure 1: Insight 1: The cache translates the interference to the attacker's observation.

3.1 Key Insights

Before formally giving our proposed model, we first introduce two key insights of cache side-channel attacks which inspire this model.

Insight 1: The root cause of all cache side-channel attacks is the existence of a path that links security-critical operations and the attacker's observations through the cache. The attacker's operation and the victim's operation interfere in the cache. The cache translates this interference to the attacker's observations. Figure 1 shows this interference.

Insight 2: Most of the cache operations have the *Local Markov Property*. We consider the cache elements, i.e. memory address, cache index and cache lines, as random variables. Local Markov Property means the probability distribution of random variables only depends on their immediate parent node/s. For example, probability of a cache line selected for eviction on a cache miss is only dependent on the cache set index, and not dependent on the memory address that mapped into this cache set, when the cache set index is given (explained further with Figure 3).

3.2 Probabilistic Information Flow Graph (PIFG) and Conditional Independence.

Information Flow Graph (IFG): This is a directed acyclic graph defined by a vertex (node) set V and an edge set E. V contains all vertices in the graph, each vertex can be interpreted as a random variable. For example, in the case of cache side-channel attacks, a vertex can represent cache index, cache line or memory address. E contains directed edges that connect two or more vertices.

Parent and child nodes: v_1 is a parent node of v_2 (thus v_2 is a child node of v_1) iff there exists a directed edge from v_1 to v_2. Note that one edge can have multiple parents but only one child.

Edge Flow Probability (EFP): is defined as the conditional probability of a child node given its parents.

Probabilistic Information Flow Graph (PIFG): This is an IFG where every edge is assigned an edge flow probability.

Edges and conditional independence: Nodes that are not connected represent information that is conditionally independent given their parent nodes.

3.3 Paths and Security-Critical Paths in PIFG

Cache side-channel attacks try to measure the time differences caused by specific memory accesses. Therefore, there are three kinds of nodes which are very special in a PIFG, viz., the **victim's security-origin nodes**, the **attacker's security-origin nodes** and the **attacker's observation nodes**. From our key insight 1, there must be a path that links the security-origin nodes and the attacker's observation nodes.

Victim's security-origin nodes represent the secure information which the attacker tries to obtain in an attack. For example, in most of the cache side-channel attacks, the victim's security-origin node is the victim's security-critical memory addresses, i.e., memory addresses that can leak security-critical information (e.g., key bits).

Attacker's security-origin nodes represent the attacker's operations (if any) required to interfere with the victim (if any). Sometimes there are no attacker's security-origin nodes in a PIFG (e.g. in cache collision attacks). An example of an attacker's security-origin node is the specific memory addresses accessed by the attacker in order to evict the victim's memory lines in the cache. It can be interpreted as the attacker's preparation for the attack.

Attacker's Observation nodes represent what can be observed by the attacker. For example, it can be the block encryption time (in an evict-and-time attack) or the memory access time (in a prime-and-probe attack) in cache side-channel attacks. Note that the attacker's observation is not limited to time (power, electromagnetic emanations, etc.), however, in this paper, we mainly discuss time-based cache side-channel attacks.

Path is a sequence of edges which connects vertices. Because PIFG is an acyclic graph, the nodes in a path are all distinct from one another.

Security-critical Path connects security-origin nodes and observation nodes. It is the path that leaks information. We define the path that connects the victim's security-origin node and the attacker's observation nodes as the victim's security-critical path, and the path that connects the attacker's security-origin node and the attacker's observation nodes as the attacker's security-critical path. If there is no attacker's security-origin node in a PIFG, there is also no attacker's security-critical path. However, in an attack, a victim's security-critical path always exists. The security-critical path is the union of the victim's security-critical path and the attacker's security-critical path.

Security-critical Nodes are the nodes on the security-critical path. We also denote with an asterisk * the nodes where secret information from a victim's security-origin node can propagate to.

3.4 A General Example of a PIFG

Figure 2 is a PIFG with:
Vertex set $V = \{A, B, C, D, E, F, G, H, I, J, K, L, M\}$
Edge set $E = \{e_1, e_2 ... e_{11}\}$
Every edge in the IFG is assigned with an edge flow probability, i.e. p_1, p_2 ... p_{11}. $p_1 = P(B|A), p_2 = P(C|B)$ and so on.
The victim's security-origin node is I, the attacker's security-origin node is A. The attacker's observation node is K.
The victim's security-critical path is $SP_V = \{e_5, e_6, e_7, e_9\}$. The attacker's security-critical path is $SP_A = \{e_1, e_4, e_6, e_7, e_9\}$. The victim's security-critical path and the attacker's security-critical path intersect at e_6.
The security-critical nodes are $SV = \{A, B, E, I, J, F, G, K\}$.

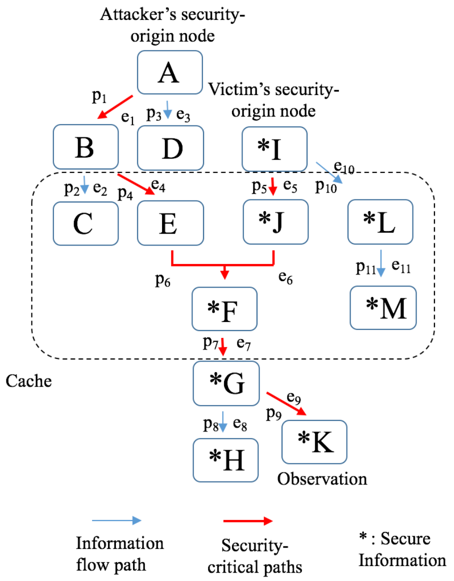


Figure 2: A general example of probabilistic information flow graph.

3.5 Use PIFG to Model Evict-and-time Attack

We now show how to model the evict-and-time attack using PIFG. Algorithm 1 shows the pseudo-code for a first-round AES evict-and-time attack. We generalize it to reveal the root causes of evict-and-time attacks, and model this class of Type 1 attacks (see Table 1) with PIFG in Figure 3.

In lines 5 through 8, we initialize the time accumulation bins and counters for each bin. There are 16 key bytes in a 128-bit AES key, and also 16 bytes in a block of plaintext to be encrypted. Each byte has 256 possible values. The attacker finds the average execution time for each of the 256 byte values, for each key byte.

In line 9, the attack performs N trials, where each trial is a block encryption of a plaintext block p (16 bytes array, shown in lines 10 through 18). p[i] is the i-th byte value of plaintext p. Rdtsc is a read-timer instruction in Intel x86 processors.

In lines 20-22, the attack accumulates the time of this block encryption in the appropriate time accumulator bin and adds one to the counter for this bin.

The 16 key-dependent AES table entries read in line 14 are $AEStable[p[i] \oplus k[i]]$. If the victim uses evicted (line 1) table entries in line 14, the encryption time $T[i][p[i]]$ will be statistically higher than if the evicted entries are never used in the victim's block encryption operation. Therefore, the attacker's observation

Algorithm 1 An example of evict-and-time attack on AES

```
1:  The attacker evicts a set $S$ of cache lines, which contains AES
    encryption table entries
2:  // Initialize time accumulation bins, T[16][256]
3:  // Initialize counters, Count[16][256]
4:  // Number of bytes in a block is 16
5:  for i = 0 to 15 do
6:      for j = 0 to 255 do
7:          T[i][j] = 0
8:          Count[i][j] = 0
9:  for n=1,2,...,N do
10:     rdtsc $t_1$
11:     // Victim encrypts plaintext block p in 10 rounds
12:     for Round r=1,2,...,10 do
13:         if r==1 then
14:             Read 16 $key-dependent$ AES table entries
15:         if $r \neq 1$ then
16:             Randomly read 16 AES table entries
17:         XOR read values with round keys
18:     rdtsc $t_2$
19:     // Attacker observes time for 1 entire block encryption, $t_2 - t_1$
20:     // Define p[i] as the i-th byte of plaintext p
21:     for i = 0 to 15 do
22:         T[i][p[i]] += $t_2 - t_1$
23:         Count[i][p[i]] ++
24: The attacker takes average, i.e. $A[i][j] = \frac{T[i][j]}{count[i][j]}$, and looks for
    a longer average in each row A[i][:].
```

is whether the block encryption time is longer than average, over a large number of trials (count[i][j]).

We generalize the root causes of evict-and-time attacks: **the attacker evicts some cache lines and the victim uses one of these evicted lines, which causes the victim's longer execution time**. Therefore, there are two "security-origin events" in this attack: one is the evicted set of cache lines S (line 1 in the example), and the other is the victim's accessed memory lines (in line 14). The attacker observes the victim's execution time (line 21). Our PIFG tries to model how these two security-origin events (in PIFG nodes) interfere in the cache and result in the attacker's observation.

In Figure 3, the security-critical edges are $\{e_1, e_2, e_3, e_4, e_5\}$. The first step in an evict-and-time attack is the attacker accessing a memory line. e_1 models the mapping between the accessed memory address and cache set index number. If the cache set index number is determined, e_2 models which cache line (in the cache set) is selected to be evicted. e_3 models the mapping between the selected cache line and the evicted memory line. Together, e_1, e_2 and e_3 model the probability that the attacker succeeds in evicting a victim's memory line out of the cache.

e_4 models whether the memory line M_E evicted by the attacker and a later memory access M_V by the victim results in a cache miss or hit. e_5 models the noise introduced in the observation. We assume the noise is Gaussian additive noise.

Based on the PIFG model for the evict-and-time attack, we assign conditional probabilities $p_1, p_2 ... p_5$ to the corresponding edges. For example, p_1 refers to the probability distribution of the cache set index number given the accessed memory address. We will show how to calculate $p_1, p_2 ... p_5$ for all secure cache architectures in Section 3.7.

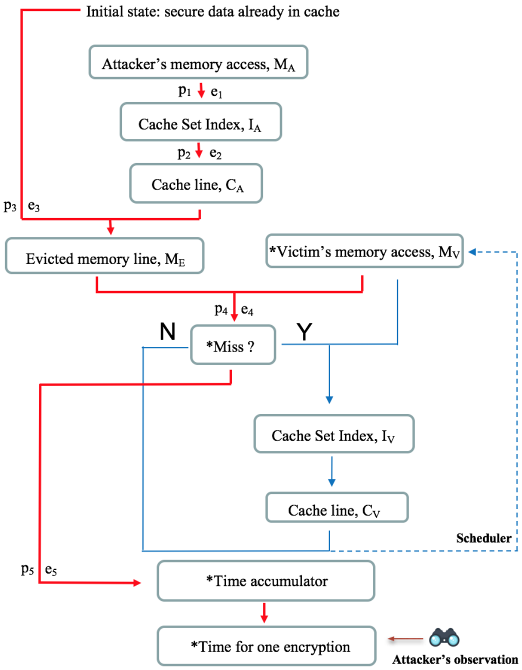


Figure 3: PIFG For Evict-and-time Attack

Table 2: Conditional probabilities model the cache architecture in PIFG.

p_n	Mappings	Cache Feature Modelled
p_1	Map from memory address to cache set	Memory to set index mapping, Cache size, associativity
p_2	Map from cache set to cache line	Replacement policy within a cache set
p_3	Map from cache line to the memory line evicted out of the cache	Line locking policy, if any
p_4	Map from evicted memory line and accessed memory line to whether the access is a hit or miss	Hit or miss policy
p_5	Map from hit/miss to access time	Noise in timing, fuzzy timer

3.6 Quantification of Attacker's Success

We have defined the PIFG model in the previous sections, now we discuss how to quantify the resistance of a secure cache architecture.

In order to quantify the attacker's success, we propose a metric, probability of attacker's success (PAS).

Probability of Attacker's Success, PAS: PAS is defined as the conditional probability of security-critical nodes (not including security-origin nodes) on the security-critical path, given the security-origin nodes. It can be interpreted as the probability that the whole attack works as the attacker desires. For example, in Figure 2, PAS is defined as:

$$\text{PAS} = P(B,E,J,F,G,K \mid A,I)$$

Lemma 1 *The joint distribution of nodes equals the multiplication of the conditional probability of nodes given their parents.*

Proof: Because PIFG is a directed acyclic graph, we can sort the nodes based on their topological order, i.e. v_i is sorted to a later position of v_j if v_i is a child node of v_j. Without loss of generality, we assume the sorted nodes are $v_1, v_2...v_n$. Because of the Local Markov Property introduced in 3.1, we have:

$$\begin{aligned}
P(v_1,v_2...v_n) &= P(v_1)P(v_2|v_1)P(v_3|v_1,v_2)...P(v_n|v_{n-1},...v_1) && (1)\\
&= P(v_1|v_1\text{'s parents})...P(v_n|v_n\text{'s parents}) && (2)\\
&= \prod_{i=1}^{n} P(v_i|v_i\text{'s parents}) && (3)
\end{aligned}$$

Note that $P(v_i|v_i^{\prime}sparents) = P(v_i)$, if v_i is a security-origin node.

Theorem 1 *PAS equals the multiplication of all edge flow probabilities on the security-critical paths.*

Proof: By definition, security-origin nodes have no parent nodes, therefore they are independent of each other. Let $o_1, o_2...o_n$ be the security-origin nodes. Let $v_1, v_2...v_m$ be the security-critical nodes which are not the security-origin nodes. By *Lemma* 1 we have:

$$\begin{aligned}
P(o_1,...o_n,v_1...v_m) &= \prod_{i=1}^{n} P(o_i|o_i\text{'s parents}) \prod_{j=1}^{m} P(v_j|v_j\text{'s parents}) && (4)\\
&= \prod_{i=1}^{n} P(o_i) \prod_{j=1}^{m} P(v_j|v_j\text{'s parents}) && (5)
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}
P(v_1,...v_m|o_1,...o_n) &= \frac{P(o_1,o_2...o_n,v_1,v_2...v_m)}{P(o_1,o_2...o_n)} && (6)\\
&= \frac{\prod_{i=1}^{n} P(o_i) \prod_{j=1}^{m} P(v_j|v_j\text{'s parents})}{\prod_{i=1}^{n} P(o_i)} && (7)\\
&= \prod_{j=1}^{m} P(v_j|v_j\text{'s parents}) && (8)\\
&= \prod_{e_i \in security-critical path} p_i && (9)
\end{aligned}$$

where p_i is the assigned edge flow probability on edge e_i.

A concrete example is shown in Figure 2, where we have

$$\begin{aligned}
\text{PAS} &= P(B,E,J,F,G,K|A,I)\\
&= P(B|A)P(E|B)P(J|I)P(F|E,J)P(G|F)P(K|G)\\
&= p_1 * p_4 * p_5 * p_6 * p_7 * p_9
\end{aligned}$$

Table 3: Conditional probability of evict-and-time attack

	p_1	p_2	p_3	p_4	p_5	PAS
SA Cache	1.0	0.125	1.0	1.0	1.0	0.125
SP Cache	0	0.125	1.0	1.0	1.0	0
PL Cache	1.0	0.125	0	1.0	1.0	0
Nomo Cache	1.0	0.167	1.0	1.0	1.0	0.167
Newcache	1.0	$1.95*10^-3$	1.0	1.0	1.0	$1.95*10^{-3}$
RP Cache	$1.56*10^-2$	0.125	1.0	1.0	1.0	$1.95*10^{-3}$
RF Cache	1.0	0.125	1.0	1.0	1.0	0.125
RE Cache	1.0	1.0	1.0	1.0	1.0	1.0
Noisy Cache	1.0	0.125	1.0	1.0	0.691	0.086

3.7 Calculate Edge Probabilities in PIFG for Evict-and-time Attack.

We use the evict-and-time attack as an example to show the calculation of conditional probabilities on each edge. We list the cache configurations we assume in our calculations in Table 4. All caches are 32 Kbytes with 64 byte lines, giving 512 cache lines. The baseline conventional cache is 8-way set-associative (64 sets) with random replacement rather than the more conventional LRU replacement. Some secure caches are also set-associative caches and all are modeled as 8-way SA with the benefit of random replacement. As discussed in Section 3.5, there are five edges on the security-critical path of an evict-and-time attack. We list the five conditional probabilities and their corresponding mappings in Table 2. We show the values of these edge probabilities in Table 3, for each cache architecture, for evict-and-time attacks, and discuss their computation below.

Table 4: Cache configurations for evaluation.

Set Associative(SA) Cache	8-way set-associative
Statically Partitioned(SP) Cache	8-way set-associative, 2 static partitions
Partition Locked(PL) Cache	8-way set-associative
Nomo Cache	$\frac{1}{4}$ ways reserved for trusted app
Newcache	512 cache lines, 1 set
Random Permutation Cache	8-way set-associative
Random Fill Cache	Window size $W_a = W_b = 64$ lines
Random Eviction Cache	Direct map, 10% random eviction
Noisy Cache	8-way set-associative, noise std $\sigma = 1$

All cache architectures except SP and RP cache have $p_1 = 1.0$. This is because they follow conventional cache mapping, with deterministic memory address to cache index mapping, so a memory address is mapped to the corresponding cache set with probability 1. However, in the case of SP cache, a memory line in the attacker's memory space cannot be mapped to a victim's cache set. That is why $p_1 = 0$ for SP cache. In RP cache, the memory address can be mapped to any cache set. Therefore, $p_1 = \frac{1}{S}$ where $S = 64$ is the number of sets.

p_2 is the conditional probability of a cache line selected for eviction (replacement) given the cache index number. Although most conventional SA caches use LRU replacement, we use random replacement for all SA caches since this gives better resilience against cache attackers. Therefore for conventional caches with random replacement policy, $p_2 = \frac{1}{W}$ where W is the number of lines in a set. Since SA, SP, PL, RP, RF and Noisy are all 8-way set-associative, $p_2 = 0.125$. For Newcache, this value is $p_2 = \frac{1}{N}$ where $N = 512$ is the total number of cache lines in the cache. Newcache can be considered as a cache with only one set. For Nomo cache, we are considering the attacker evicts an unreserved cache line, thus $p_2 = \frac{1}{W'}$ where W' is the number of unreserved cache lines in a cache set. This is $8 * \frac{3}{4} = 6$ ways. RE is a Direct-mapped cache, so $W = 1$, and $p2 = \frac{1}{W} = 1$.

p_3 models the probability distribution of the memory line being evicted given a selected cache line. This conditional probability is 1.0 for all cache architectures except PL cache. In all cache architectures except PL cache, the memory line in the selected cache line must be evicted out with probability 1. However, in PL cache, if the cache line which contains the security-critical data is correctly locked, the memory line will not be evicted out.

p_4 models whether the victim's access is a hit or miss, given the memory line evicted out in the previous step and the victim's accessed memory address. Unfortunately, no secure cache architectures play any tricks in this step. The victim's access must be a miss if the memory line is evicted out in the previous step and accessed again by the victim. **Our model suggests that there may be a new opportunity for a new secure cache design**.

We assume Gaussian noise in the attacker's observation on edge e_5, i.e the time distribution for a hit is a Gaussian distribution located at 0, and the time distribution for a miss is also a Gaussian distribution but located at 1. The Gaussian assumption follows the Central Limit Theorem, and we have done experiments on measuring real cache hit/miss time to verify it. We use two types of errors, False Positives (FP) and False Negatives (FN) as metrics. The illustration of FP and FN are also shown in the Figure 4. Let σ be the normalized standard deviation, we have:

$$P(\text{Attacker errors}) = 1 - \frac{1}{\sqrt{\pi}} \int_{\frac{1}{2\sqrt{2}\sigma}}^{\infty} e^{-t^2} dt = 1 - \frac{1}{2} erfc(\frac{1}{2\sqrt{2}\sigma})$$

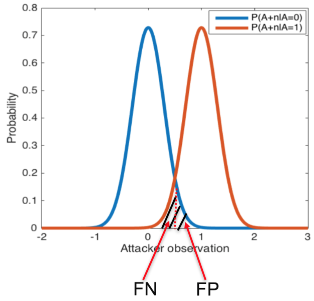


Figure 4: p5 values of evict-and-time attack

The multiplication of all these probabilities for a given cache architecture gives the Probability of Attack Success (PAS) in the last column of Table 3. PAS of 1, or close to 1, means the cache (in this row) is not resilient to this class of attacks. PAS of 0, or close to 0, indicates the cache is resilient to this class of attacks. SP and PL caches are definitely resilient. RP and Newcache also have small PAS values, and Newcache actually has higher resilience than RP cache, if we also consider the "pre-PAS" initialization probabilities in section 5.

4 MODEL OTHER ATTACKS WITH PIFG

In Section 3, we introduced our general probabilistic information flow graph model and used it for quantifying evict-and-time attacks. In this section, we first show that our proposed PIFG model is very extensible to model new attacks and new architectures by using the cache collision attack (Type 3) as an example. We also show the models for Types 2 and 4 cache side-channel attacks.

4.1 Cache Collision Attack

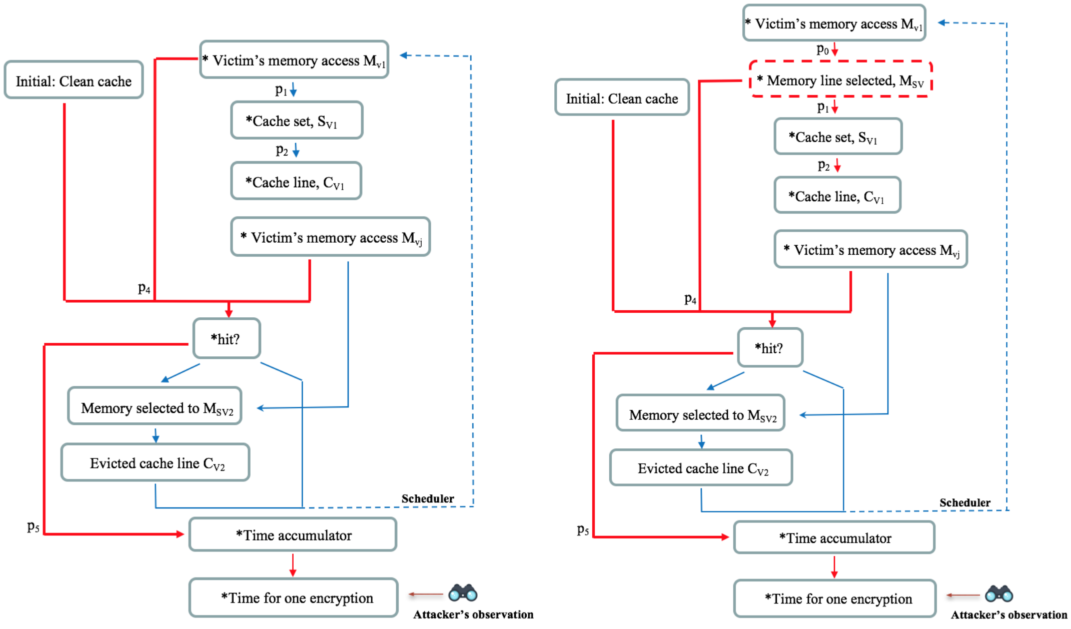
The cache collision attack is special because the attacker does not need to interfere with the victim. We build our initial PIFG model for collision attack in Figure 5 (a). There are only two edges, p_4 and p_5, on the security-critical path, and before RF cache was introduced, no secure caches do good defenses in these two steps. p_4 is the probability of a hit on a second memory access to the same cache line as the first memory access. This is 1.0 for conventional cache architectures because if the first memory access fetches a memory line into the cache, the second access of that memory line must be a hit. This is, in fact, the main purpose of a cache. (We assume there is no self-eviction in the victim's security-critical accesses.)

In RE cache, $p_4 = 1 - \frac{1}{NT}$, where N is the number of cache lines in the cache and T is the interval between two random evictions. Intuitively, it means the probability that "the later memory access is a hit" equals the probability that "the previously fetched memory line is not evicted because of the random evictions in RE cache". In our cache configuration, $N = 512$ cache lines and $T = 10$ memory accesses. Similar to the discussion in evict-and-time attack, p_5 is 1.0 for all but the noisy cache (0.691).

It seems that these secure cache architectures cannot defend against cache collision attacks, because getting a cache hit from a previous memory access is the basis of the demand fetch policy used in all caches (except RF cache). Random Fill (RF) cache is specially designed to defend against cache collision attacks. We will show how to include RF cache into our model. RF cache randomly selects a memory in a nearby window of the accessed memory line, and fetches that selected memory line into the cache. In order to model this feature, we need to add one more node, i.e. selected memory line, in the graph (Figure 5 (b) dotted node).

p_0 models the conditional probability of the selected memory line brought into the cache given the accessed memory line. $p_0 = 1.0$ for conventional cache architectures and all the other cache architectures considered so far (except for RF cache), because the selected memory line is the accessed memory line in all these caches due to the demand fetch policy. $p_0 = \frac{1}{W_a+W_b+1}$ for RF cache, where W_a and W_b are backward and forward window sizes respectively, i.e. the fetched memory line is randomly selected in the window $[M_{V1} - W_a, M_{V1} + W_b]$. In our cache configuration, we choose $W_a = W_b = 64$, which covers the entire AES encryption tables.

From the PAS results for cache collision attacks in Table 5, we see that RF cache is the only cache that can defeat collision attacks.



(a) Old model for collision attack

(b) New model for collision attack

Figure 5: Add a node in PIFG model to model new cache architecture, RF, for cache collision attacks.

Although the absolute value of PAS for RF cache does not look very small (7.75 ∗ 10−3), it is not higher than the probability that the attacker just makes a random guess, where each cache line containing AES table entries is equally likely to be brought in. Hence, our model classifies RF cache as resilient against cache collision attacks.

Table 5: PAS of cache collision attacks.

	p_0	p_4	p_5	PAS
SA Cache	1.0	1.0	1.0	1.0
SP Cache	1.0	1.0	1.0	1.0
PL Cache	1.0	1.0	1.0	1.0
Nomo Cache	1.0	1.0	1.0	1.0
Newcache	1.0	1.0	1.0	1.0
RP Cache	1.0	1.0	1.0	1.0
RF Cache	$7.75 * 10^{-3}$	1.0	1.0	$7.75 * 10^{-3}$
RE Cache	1.0	0.9998	1.0	0.9998
Noisy Cache	1.0	1.0	0.691	0.691

4.2 Prime-and-probe Attack

We show the prime-and-probe attack model in Figure 6. In general, the prime-and-probe attack consists of three memory-access phases: (A) The attacker primes (evicts) victim cache lines, (B) The victim misses on his accessed memory line, and evicts the attacker's memory line in (A), and (C) The attacker accesses the same memory line in (A) and misses on that. p_{11}, p_{21}, p_{31} model the first memory access phase. This phase is the same as for the evict-and-time attack, because the only goal of this phase is evicting the victim's data out of the cache. This shows how the similarity in the evict-and-time attack and prime-and-probe attack are reflected in our model. p_{12}, p_{22}, p_{32} model the second memory access phase. p_{42} models the third memory access phase. Different from phases (A) and (B) which require either a specific memory line to be fetched into the cache (A), and a specific memory line to be evicted out (B), the attacker wins in this phase as long as the attacker misses on his memory access. Therefore, no cache architectures do any trick in p_{42}. p_5 is the same as p_5 in the evict-and-time attack, modeling the noise introduced. Only the noisy cache tries to mess the attacker's observation in p_5.

The PAS values for these Type 2 (prime-and-probe) attacks are smaller than for the Type 1 attacks for the same cache. That is because both attacks require the attacker evicting (priming) the cache. However, in the victim's memory access phase, the Type 1 attack only requires a cache miss, but a Type 2 attack also requires the victim to evict a specific (attacker's) memory line. Moreover, the Type 2 attack requires one more cache miss in phase (C).

4.3 Flush-and-reload Attack

We show the flush-and-reload attack model in Figure 7. In general, flush-and-reload attacks are very similar to cache collision attacks except the collision is the attacker's memory access M_A (in the Reload step) with the victim's memory access M_{sv} to a shared memory line.

Here, p_0 models the probability of the selected memory line in an RF cache window, given the accessed memory line. It is the same as in the cache collision attack.

p_4 models the probability of an attacker's memory access causing a cache hit given the line has been fetched by the victim, and models

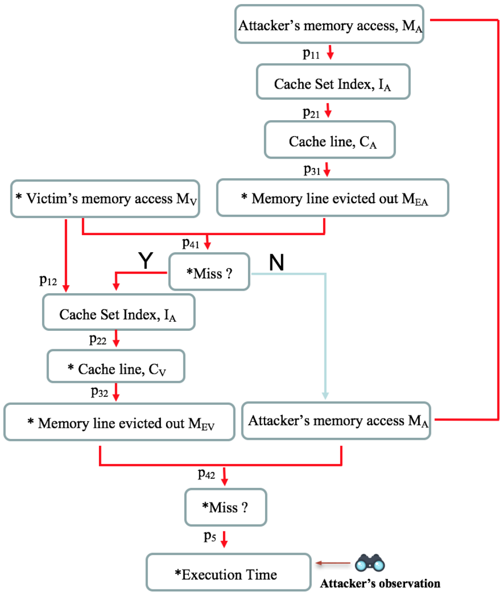


Figure 6: Information Flow of Prime-and-probe Attack.

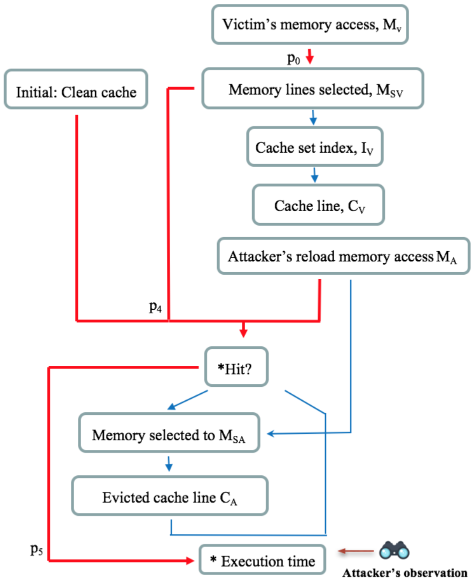


Figure 7: Information Flow of Flush-and-reload Attack.

the difference between cache-collision attacks and flush-and-reload attacks. Different from cache collision attacks, the probability $p_4 = 0$ for Newcache and RP cache because their PID (process identifier) feature ensures that the shared data of different processes with different PIDs will result in different cache line tags, so the data fetched by the victim will not cause a hit for the attacker's memory access.

p_5 is the same as previous cache side-channel attacks that model the noise introduced in the attacker's observation.

Finally, we list PAS for all nine cache architectures against four types of attacks in Table 6. The cache is more resilient if the number is smaller. Note that when considering multiple line evictions, the randomization based secure caches can have even lower PAS.

Table 6: PAS of four types of attacks for 9 cache architectures

	Type 1	Type 2	Type 3	Type 4
SA Cache	0.125	$1.56*10^{-2}$	1.0	1.0
SP Cache	0	0	1.0	1.0
PL Cache	0	0	1.0	1.0
Nomo Cache	0.167	0	1.0	1.0
Newcache	$1.95*10^{-3}$	$3.80*10^{-6}$	1.0	0
RP Cache	$1.95*10^{-3}$	$3.80*10^{-6}$	1.0	0
RF Cache	0.125	$1.27*10^{-4}$	$7.75*10^{-3}$	$7.75*10^{-3}$
RE Cache	1.0	1.0	0.9998	0.9998
Noisy Cache	0.086	0.012	0.691	0.691

5 PREREQUISITE OF ATTACKS

In the collision attack and flush-and-reload attack in Section 4, we assume a clean cache before the attack is launched. However, there is still a missing puzzle: ***how can the attacker clean the cache in advance if the security-critical data has been prefetched***. In practice, the attacker has two options: a) use special instructions in specific architectures, e.g. clflush in x86, or b) access the memory many times to evict out the security-critical data. The first option is architecture-specific and sometimes requires special privilege, therefore in the following discussion, we only analyze the probability of an attacker successfully cleaning the cache with large numbers of memory accesses. This cleaning step is the prerequisite of collision and flush-and-reload attacks. We discuss two replacement policies for each cache architecture: Least Recently Used (LRU) and random.

For fairness in the comparison, we define that an attacker is successful in cleaning the cache if he is able to remove all data in a cache set and fill it with his own data. We analyze the probability of an attacker's preparation success (pre-PAS) under different cache architectures below.

A: Set-associative (SA) Cache

In a conventional SA cache with LRU replacement, an attacker is guaranteed to occupy the whole cache set after the number of trials (memory accesses) reaches the set-associativity. For example, in an 8-way set-associative LRU cache, an attacker can fill up a whole cache set with 8 or more memory accesses. Hence, in a w way set-associative cache:

$$pre\text{-}PAS_{SA\_LRU} = \begin{cases} 0 & k < w \\ 1 & k \geq w \end{cases} \tag{10}$$

We model the random replacement based cache as a ball-picking game. Assume the cache's associativity is w and the attacker can access at most k memory lines. In the ball-picking game, there are w balls (cache lines) in a box (cache set). The attacker tries k times, and in each trial (memory access) he randomly chooses a ball from the box and then puts it back. The attacker succeeds if he picks all balls (accesses all lines in a set) at least once, and fails if there exists a ball that he never picks (a memory line he never accesses) in k

trials. The pre-PAS of w way set-associative cache with k trials is:

$$pre\text{-}PAS_{SA\_rand} = \begin{cases} 0 & k < w \\ \sum_{i=1}^{w} (-1)^{i+w} C_w^i (1 - \frac{i}{w})^k & k \geq w \end{cases} \tag{11}$$

B: Newcache

We define that an attacker succeeds if he can evict a target physical cache line in Newcache. The probability of cleaning a specific cache line in Newcache is:

$$pre\text{-}PAS_{Newcache} = 1 - (1 - \frac{1}{2^n})^k$$

C: Static Partitioning (SP) Cache and Partition Locked (PL) Cache

SP cache and PL cache are both partitioning based secure caches. In SP cache, ideally the attacker and the victim do not share partitions. In this case, it is impossible for the attacker to evict a whole target cache set, i.e. $pre\text{-}PAS_{SP} = 0$. However, PL cache requires secure critical data to be stored in advance before encryption or other secure tasks, i.e. it requires prefetching. If secure data are not prefetched into the cache, PL cache performs just like a conventional set-associative cache [13].

D: Random Permutation Cache (RP) Cache

Random Permutation (RP) cache assigns each secure domain a different permutation table. However, nothing prevents the attacker from disabling this permutation feature of his own process. In this situation, an RP cache is the same as an SA cache to the attacker.

E: Random Fill (RF) Cache

Now we consider a Random Fill (RF) cache based on an SA cache. Random fill cache cannot prevent an attacker from evicting a deterministic set, because the attacker can set his window size equal to 0. In this situation, RF cache degrades to an SA cache. This is very similar to RP cache, where an attacker is also able to avoid the security mechanisms and launch attacks.

F: Random Eviction (RE) Cache

The random eviction operation in Random Eviction cache [5] is designed to randomly and periodically evict a cache line. We find that, in the cleaning phase, the random eviction mechanism actually helps the attacker. The random evictions induced by RE cache can be considered as "free lunches" from the attacker's point of view.

Assume the interval between two random evictions is N, i.e. a random eviction happens every N memory accesses. Therefore, the attacker can get $\lfloor \frac{k}{N} \rfloor$ "free lunches" after k memory accesses. This is equivalent to $k + \lfloor \frac{k}{N} \rfloor$ evictions. Pre-PAS of RE cache is shown below:

$$pre\text{-}PAS_{RE\_LRU} = \begin{cases} 0 & k + \lfloor \frac{k}{N} \rfloor < w \\ 1 & k + \lfloor \frac{k}{N} \rfloor \geq w \end{cases}$$

$$pre\text{-}PAS_{RE\_rand} = \begin{cases} 0 & k + \lfloor \frac{k}{N} \rfloor < w \\ \sum_{i=1}^{w} (-1)^{i+w} C_w^i (1 - \frac{i}{w})^{k + \lfloor \frac{k}{N} \rfloor} & k + \lfloor \frac{k}{N} \rfloor \geq w \end{cases}$$

G: Nomo Cache

Nomo Cache reserves a portion α of lines in each cache set for the security-critical process. If the reserved cache lines are sufficient for the secure process, i.e. the secure process uses no more than αw lines in any cache set, Nomo cache thoroughly separates the attacker's cache space and the victim's cache space.

If the reserved cache lines are not enough for the victim's secure process, the attacker can interfere with the victim by first filling up shared lines with his own data. We define the attacker succeeds if he can evict all **shared** lines. Formally, assume the maximum number of lines used by the victim's process in a set is M, we have:

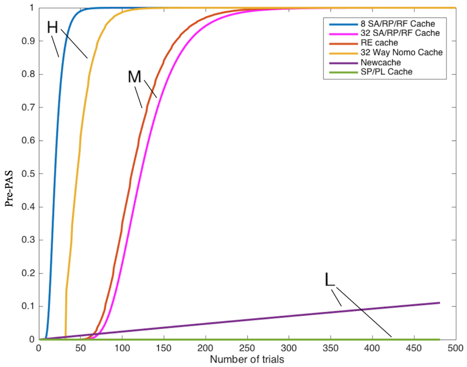


Figure 8: Pre-PAS of different cache architectures with random replacement policy.

If $M \leq \alpha w$:

$$pre\text{-}PAS_{Nomo} = 0$$

If $M \geq \alpha w$:

$$pre\text{-}PAS_{Nomo\_LRU} = \begin{cases} 0 & k < (1-\alpha)w \\ 1 & k \geq (1-\alpha)w \end{cases}$$

$$pre\text{-}PAS_{Nomo\_rand} = \begin{cases} 0 & k < (1-\alpha)w \\ \sum_{i=1}^{(1-\alpha)w} (-1)^{i+(1-\alpha)w} C_{(1-\alpha)w}^i (1 - \frac{i}{(1-\alpha)w})^k \\ & k \geq (1-\alpha)w \end{cases}$$

As expected, when $\alpha = 0$, Nomo cache degrades to a conventional set-associative cache.

We show results of pre-PAS for different cache architectures with random replacement policy in Figure 8. A cache architecture with pre-PAS=L is more resilient to attacks which require cleaning the cache. Hence, for cache architectures with pre-PAS=L, extra security can be provided for Type 3 and 4 attacks.

In Figure 8, we label the caches as having High (H), Medium (M) or Low (L) pre-PAS values. 8-way SA, RP, RF and Nomo caches have pre-PAS=H. RE and 32-way SA, RP and RF caches have pre-PAS=M. Newcache, SP and PL caches have pre-PAS = L.

We make some interesting observations from this figure:

- Pre-PAS increases as the number of attacker's trials increases. Pre-PAS decreases as associativity increases.
- None of the curves reaches 1.0 even though some of them are very close. This indicates that no matter how many accesses have been done by the attacker, he cannot be 100% sure that the cache is clean, with random replacement.
- Partitioning based caches have pre-PAS=0. Newcache performs the best among all randomization based cache architectures and has a very low pre-PAS.

We should not read too much into small differences in PAS values between different caches. Rather, we recommend using the combination of PAS and pre-PAS values to give broad categories of

Table 7: Resilience of caches to cache side channel attacks. $'\surd'$ = high resilience, $'X'$ = low resilience.

	Type 1	Type 2	Type 3	Type 4
SA Cache	X	X	X	X
SP Cache	√	√	X	X
PL Cache	√	√	X	X
Nomo Cache	X	√	X	X
Newcache	√	√	X	√
RP Cache	√	√	X	√
RF Cache	X	√	√	√
RE Cache	X	X	X	X
Noisy Cache	X	X	X	X

the relative security (i.e., resilience) of different cache architectures against each type of cache attacks, as illustrated by Table 7. Here, a "√" indicates high resilience to that class of cache attacks, corresponding to PAS and pre-PAS values of 0 or close to 0. An "X" indicates low (or no) resilience, where the PAS value is close to (or equal to) 1.

6 VALIDATION OF PIFG MODEL

Since there are no machines that have been built with any of the 8 secure architectures proposed, we cannot compare our results with real hardware. In this section, we confirm that our results agree with the simulation results available for each cache architecture. We illustrate this below.

Case Study: Type 1 attacks (e.g., Evict-and-time)

Intuitively, partitioning-based secure caches make it impossible for the attacker to succeed if his attack involves interfering with the victim's cache usage, because he cannot evict any cache lines in the victim's partition. This corresponds to '√' in Table 7 for SP and PL caches for Type 1 and Type 2 attacks. The results for randomization based caches are less obvious. Fortunately, Liu [17] showed the comparison of Newcache and conventional SA cache, replicated in Figure 9. The conventional SA cache on the left has significant high points (longer execution time) near x=0; however, Newcache (right) does not. This shows that Newcache is more resilient than conventional SA cache in defending against Type 1 attacks, which is consistent with our PIFG results.

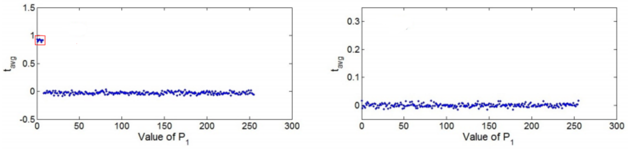
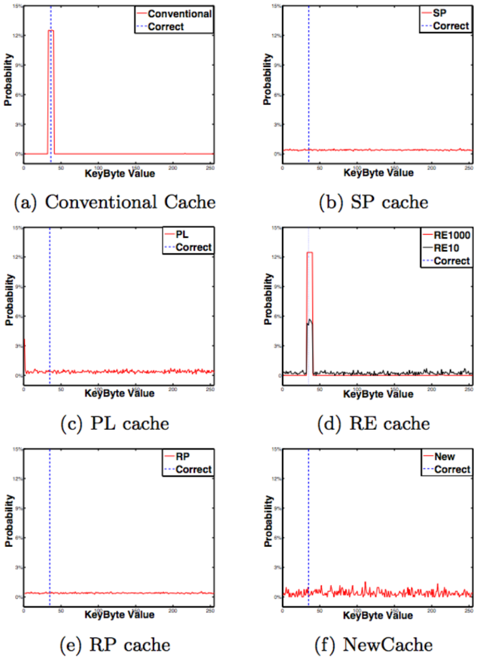


Figure 9: Simulation results of Type 1 attacks, from [17]. Left: Conventional 8-way SA cache. Right: Newcache.

Case Study: Type 2 attacks (e.g., Prime-and-probe)

Zhang [35] simulates a conventional SA cache and five secure cache architectures against Type 2 attacks, replicated in Figure 10. Solid red lines are the candidate keys' probability distribution, dotted blue lines are the key byte values. A flat red line means no key leakage, while overlap of red and blue lines shows key leakage. We see that SP, PL, Newcache and RP caches can defend against type 2 attacks ('√' in Table 7), while conventional SA and RE caches leak key information ('X' in Table 7).



(a) Conventional Cache

(b) SP cache

(c) PL cache

(d) RE cache

(e) RP cache

(f) NewCache

Figure 10: Simulation results of Type 2 attacks, from [35].

7 CONCLUSIONS AND FUTURE WORK

We propose a new Probabilistic Information Flow Graph (PIFG) model to show the root causes of information leakage. We then propose a new metric, Probability of Attacker's Success (PAS), to quantify a cache architecture's resilience against cache side-channel attacks. It is a unified model which can be applied to all types of cache side-channel attacks and different cache architectures. We use this model to evaluate nine caches' resilience (security) to all four types of cache sie-channel attacks. Our model is very extensible and can model new attacks and new cache architectures.

We have provided an unbiased and uniform way to simultaneously model the attacker, the victim and the cache architecture in the design phase. We hope that this helps advance the goal of more rigorous security analysis of secure cache architectures, in particular, and secure hardware-software architectures, in general.

Some directions for future work we suggest are: applying our PIFG model for software or system level defenses, applying it to other side-channel attacks, such as memory buses, EM and power consumption side channels, and providing a tool for computing PAS.

Acknowledgements

This work was supported in part by NSF SaTC 1526493 and SRC 2015-TS-2632, and an Intel research gift. We thank Fangfei Liu for very useful discussions on the modeling and clarifications on Random Fill cache designs.

REFERENCES

[1] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. 2010. Acoustic Side-Channel Attacks on Printers.. In *USENIX Security symposium*. 307–322.

[2] Daniel J Bernstein. 2005. Cache-timing attacks on AES. (2005).

[3] Joseph Bonneau and Ilya Mironov. 2006. Cache-collision timing attacks against AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 201–215.

[4] Ernie Brickell, Gary Graunke, Michael Neve, and Jean-Pierre Seifert. 2006. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. *IACR Cryptology ePrint Archive* 2006 (2006), 52.

[5] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: a metric for measuring information leakage. *ACM SIGARCH Computer Architecture News* 40, 3 (2012), 106–117.

[6] Leonid Domnitser, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2010. A predictive model for cache-based side channels in multicore and multithreaded microprocessors. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer, 70–85.

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

[8] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache Games–Bringing Access-Based Cache Attacks on AES to Practice. In *2011 IEEE Symposium on Security and Privacy*. IEEE, 490–505.

[9] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. 2010. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy* 8, 6 (2010), 40–47.

[10] Naofumi Homma, Takafumi Aoki, and Akashi Satoh. 2010. Electromagnetic information leakage for side-channel analysis of cryptographic modules. In *2010 IEEE International Symposium on Electromagnetic Compatibility*.

[11] Emilia Käsper and Peter Schwabe. 2009. Faster and timing-attack resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 1–17.

[12] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. 2012. STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 189–204.

[13] Jingfei Kong, Onur Aciicmez, Jean-Pierre Seifert, and Huiyang Zhou. 2008. Deconstructing new cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 2nd ACM workshop on Computer security architectures*. ACM, 25–34.

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

[15] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. 2012. Automatic quantification of cache side-channels. In *International Conference on Computer Aided Verification*. Springer, 564–580.

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

[17] Fangfei Liu and Ruby B Lee. 2013. Security testing of a secure cache design. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 3.

[18] Fangfei Liu and Ruby B Lee. 2014. Random fill cache architecture. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 203–215.

[19] Fangfei Liu, Hao Wu, Kenneth Mai, and Ruby B Lee. 2016. Newcache: Secure Cache Architecture Thwarting Cache Side-Channel Attacks. *IEEE Micro* 36, 5 (2016), 8–16.

[20] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*. 605–622.

[21] Stefan Mangard. 2002. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *International Conference on Information Security and Cryptology*. Springer, 343–358.

[22] Kouhei Nadehara, Masao Ikekawa, and Ichiro Kuroda. 2004. Extended instructions for the AES cryptography and their efficient implementation. In *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*. IEEE, 152–157.

[23] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. 2005. A side-channel analysis resistant description of the AES S-box. In *International Workshop on Fast Software Encryption*. Springer, 413–423.

[24] Dan Page. 2005. Partitioned Cache Architecture as a Side-Channel Defence Mechanism. *IACR Cryptology ePrint Archive* 2005 (2005), 280.

[25] Colin Percival. 2005. Cache missing for fun and profit. (2005).

[26] Chester Rebeiro and Debdeep Mukhopadhyay. 2012. Boosting profiled cache timing attacks with a priori analysis. *IEEE Transactions on Information Forensics and Security* 7, 6 (2012), 1900–1905.

[27] François-Xavier Standaert, Tal G Malkin, and Moti Yung. 2009. A unified framework for the analysis of side-channel key recovery attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 443–461.

[28] Stefan Tillich and Johann Großschädl. 2006. Instruction set extensions for efficient AES implementation on 32-bit processors. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 270–284.

[29] Eran Tromer, Dag Arne Osvik, and Adi Shamir. 2010. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology* 23, 1 (2010), 37–71.

[30] Bhanu C Vattikonda, Sambit Das, and Hovav Shacham. 2011. Eliminating fine grained timers in Xen. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 41–46.

[31] Zhenghong Wang and Ruby B Lee. 2007. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*, Vol. 35. ACM, 494–505.

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

[33] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. 2011. An exploration of L2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 29–40.

[34] Yuval Yarom and Katrina Falkner. 2014. Flush+ reload: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. 719–732.

[35] Tianwei Zhang and Ruby B Lee. 2014. New models of cache architectures characterizing information leakage from cache side channels. In *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 96–105.

[36] Tianwei Zhang, Fangfei Liu, Si Chen, and Ruby B Lee. 2013. Side channel vulnerability metrics: the promise and the pitfalls. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2.

[37] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 305–316.