Metior: A Comprehensive Model to Evaluate Obfuscating Side Channel Defense Schemes

Peter W. Deutsch
MIT
Cambridge, MA, USA
pwd@mit.edu

Weon Taek Na
MIT
Cambridge, MA, USA
weontaek@mit.edu

Thomas Bourgeat
MIT
Cambridge, MA, USA
bthom@mit.edu

Joel S. Emer
MIT/NVIDIA
Cambridge, MA, USA
jsemer@mit.edu

Mengjia Yan
MIT
Cambridge, MA, USA
mengjiay@mit.edu

ABSTRACT

Microarchitectural side-channels enable an attacker to exfiltrate information via the observable side-effects of a victim's execution. *Obfuscating* mitigation schemes have recently gained in popularity for their appealing performance characteristics. These schemes, including randomized caches and DRAM traffic shapers, limit, but do not completely eliminate, side-channel leakage. An important (yet under-explored) research challenge is the quantitative study of the security effectiveness of these schemes, identifying whether these obfuscating schemes help increase the security level of a system, and if so, by *how much*.

In this paper, we address this research challenge by presenting Metior, a comprehensive model to quantitatively evaluate the effectiveness of obfuscating side-channel mitigations. Metior offers a way to reason about the *flow of information* through obfuscating schemes. Metior builds upon existing information theoretic approaches, allowing for the comprehensive side-channel leakage evaluation of active attackers, real victim applications, and state-of-the-art microarchitectural obfuscation schemes. We demonstrate the use of Metior in the concrete leakage evaluation of three microarchitectural obfuscation schemes (fully-associative random replacement caches, CEASER-S, and Camouflage), identifying unintuitive leakage behaviours across all three schemes.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**.

KEYWORDS

hardware security, side-channels, leakage quantification

ACM Reference Format:
Peter W. Deutsch, Weon Taek Na, Thomas Bourgeat, Joel S. Emer, and Mengjia Yan. 2023. Metior: A Comprehensive Model to Evaluate Obfuscating Side-Channel Defense Schemes. In *Proceedings of the 50th Annual International Symposium on Computer Architecture ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3579371.3589073

1 INTRODUCTION

Side-channel attacks, a class of attacks which exploit shared microarchitectural state between a victim and an attacker, present a large threat to the security of shared computing systems. Through such attacks, an attacker can steal secrets from a victim by observing visible contention on shared microarchitectural structures [41], including caches [12, 29, 32, 50], memory controllers [45], on-chip networks [9, 47], and functional units [1].

A common strategy to mitigate microarchitectural side-channel attacks is through *obfuscation*, altering the victim's observable microarchitectural footprint to make it more difficult (but not impossible) for an attacker to ascertain the victim's secrets. Examples of mitigation schemes employing obfuscation include randomly mapped caches [35, 36, 48], memory traffic obfuscation schemes [55], and degrading attacker timing granularities [31, 43].

While such schemes rightfully claim that they improve the security of computing systems, their security evaluation has presented a unique challenge to the computer architecture community. There is a large body of work that provides *boolean* measures of security, evaluating the presence or absence of leakage within a design [3, 13, 18]. Since obfuscating defense schemes admit some amount of leakage, boolean measures of security are not particularly useful in these schemes' evaluation. Instead, we are interested in *quantifying* the amount of information an attacker can glean after the victim's accesses are obfuscated by the scheme under study. This leakage is highly dependent on a wide variety of factors, including the internal behaviours of the obfuscation scheme, the attacker's strategy, and the victim's microarchitectural footprints. Leakage quantification helps us reveal *how leaky* certain obfuscation techniques are, which attack strategies maximize leakage, and which applications are most vulnerable.

This paper presents Metior, a model to quantitatively evaluate the security of microarchitectural obfuscating defense schemes. Metior offers an extensible model that can be used to study the behaviour of a wide range of microarchitectural obfuscation schemes, such as those protecting caches and DRAM buses, rather than limiting itself to one specific scheme. To drive the concrete quantification of leakage and account for the non-determinism of obfuscating schemes, Metior encapsulates the victim and attacker's access patterns using a unified random variable model. Our formulation is inspired by Issa et al. [49], significantly expanding the analysis scope of prior work to cover a broad range of microarchitectural channels and active attackers. Using random variables, Metior converts the detailed study of the interactions between the attacker and the victim's access patterns into a mathematical exercise. Metior leverages the Maximal Leakage metric [49] to describe an attacker's ability to leak information about the victim's access patterns compared to a blind guess. Doing so enables Metior to comparatively evaluate how the effectiveness of their schemes varies across differing defense parameters, victims, and attackers, helping mitigation designers better understand their schemes prior to deployment.

We demonstrate the use of Metior through the security evaluation of three state-of-the-art obfuscation schemes. First, we study the defense efficacy of fully-associative random replacement caches when protecting AES, a program with a large secret space, against cache occupancy attackers [40]. We then expand our analysis to broadly consider different classes of attacks, including Probabilistic Prime+Probe [34] and cache occupancy attacks, mounted on Skewed-CEASER [35]. Finally, we evaluate the leakage of Camouflage [55], an obfuscating scheme which protects against DRAM timing attacks.

Via these case studies, Metior has produced several insights regarding the properties of these state-of-the-art obfuscation schemes. Here we highlight one of these findings: prior work [2] has attributed the success of Probabilistic Prime+Probe attacks (PPP) to targeted cache set-conflicts. Metior finds that, under certain configurations, PPP attacks primarily exploit occupancy effects. This finding demonstrates how Metior can be used to reveal new insights into how randomized caches mitigate different attacks.

2 BACKGROUND

2.1 Side-Channel Attacks

Microarchitectural side-channel attacks leak secret information from a victim to an attacker through the visible side-effects of a victim's execution. As first described in DAWG [24] and CaSA [2], this flow of information can be described through a telecommunications analogy. The victim executes code which operates on secret information, utilizing a shared microarchitectural resource (the *channel*). The victim acts as a *transmitter*, with its secret-dependent actions leaking information by *modulating* (i.e., changing of the state of) this channel. These modulations are received by the attacker through its observations. Once the attacker has observed the victim's modulations, it can then attempt to decode the secret.

2.2 Classifications of Side-Channels

To better understand the scope of Metior, we provide a brief classification of microarchitectural side-channel attacks. Microarchitectural side-channels can broadly be categorized to be either *persistent* or *ephemeral*. Further, attackers who leverage such side-channels can be classified to be either *active* or *passive*.

Persistent vs. Ephemeral Side-Channels[1]**.** In a *persistent* side-channel, the attacker attempts to observe lasting state changes in the channel caused by the victim's actions. In particular, these state changes are not self-resetting and last until another microarchitectural event modifies them. Classic cache attacks, such as Prime+Probe [29] and Flush+Reload [50], are examples of attacks that exploit persistent side-channels.

Conversely, *ephemeral* side-channels rely on transitory state changes in the channel that are self-resetting and become unobservable after a short period of time. To observe these ephemeral state changes, an attacker must modulate the channel at the same time as the victim. An example of an attack class exploiting ephemeral side-channels is memory controller timing attacks [45], wherein an attacker attempts to observe a victim's access patterns to a shared DRAM memory controller. Each of the victim's requests only occupies the resources inside the memory controller for a short period of time, after which no contention can be detected. An attacker must thus simultaneously issue requests from a different core to collide with the victim's requests, observing the delays imposed onto the timing of its own requests.

Active vs. Passive Attackers. Side-channel attackers can further be classified to be either *passive* or *active*, depending on whether the attacker and the victim's actions interfere with each other or not during an attack. *Passive* attackers directly observe the victim's microarchitectural footprint without having any effect on it, such as physically measuring the power consumption or the termination time of a victim program. Conversely, *active* attackers modulate the channel to obtain their observations, and thus unavoidably change the microarchitectural state and affect the victim's activities. For example, in cache attacks like Prime+Probe, the attacker actively modifies the state of the cache to facilitate its observations. Similarly, in memory controller attacks, the attacker actively issues requests which cause contention, affecting any other requests that use the controller.

2.3 Side-Channel Mitigation Schemes

A wide variety of side-channel mitigation schemes exist in the literature, achieving security from orthogonal approaches. For instance, side-channel leakage can be mitigated at the application level by writing programs that do not modulate the channel differently for different secrets, such as constant-time approaches [7]. Side-channel mitigations also include defenses against specific attack classes, such as speculative execution attacks (e.g. Spectre [25] and Meltdown [28]). In this paper, we focus on mitigation schemes that can block both speculative and non-speculative side-channel leakage. Such schemes can be classified into two categories: *partitioning* and *obfuscating*, each with varying security goals and performance implications.

Partitioning schemes aim to offer strict isolation guarantees, ensuring that any modulation by a transmitter can never be observed by a receiver. Partitioning schemes generally guarantee a non-interference property [52], ensuring that modulations from differing security domains cannot influence each other. Examples

[1]"Persistent" and "Ephemeral" side-channels are sometimes referred to as "Stateful" and "Stateless" side-channels respectively. However, we note that both side-channels rely on modifications to microarchitectural states.

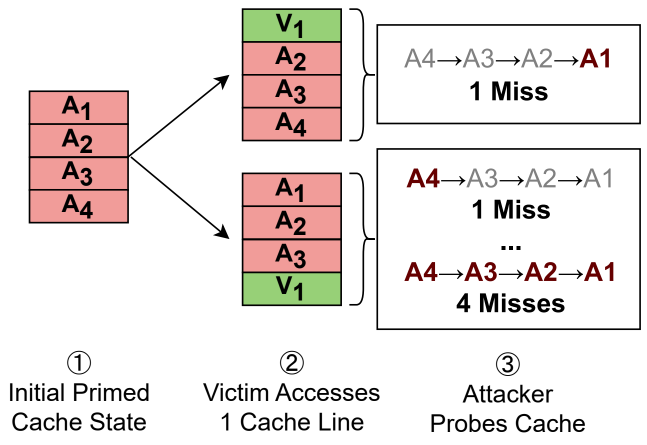

Figure 1: A motivating example to study random replacement policy as a simple obfuscating scheme. The example shows the probabilistic nature of obfuscating schemes. Red blocks are attacker accessed, green blocks are victim accessed.

of such schemes include spatially partitioned caches [15, 24, 46] and temporally partitioned DRAM scheduling schemes such as Fixed Service [38]. The security of partitioning schemes has been well studied with formal tools [3, 4, 13, 17, 18, 42].

Another class of mitigation schemes implements techniques to obfuscate the victim's microarchitectural activities. These obfuscation schemes limit, but do not completely eliminate, the attacker's ability to leak side-channel information through microarchitectural observations. For instance, randomly mapped caches, including Skewed-CEASER [35], ScatterCache [48], and MIRAGE [36], use randomization to obfuscate the set mappings of the cache, making it difficult for an attacker to glean information through collisions. Other schemes such as Camouflage [55] aim to reduce leakage from ephemeral attacks by shaping the timing of a victim's requests to a fixed distribution.

3 PROBLEM FORMULATION

There exists a critical research problem in the adoption of obfuscation schemes for the mitigation of side-channel leakage; that is, the security properties of these schemes has not been well understood. Applying boolean measures of security to evaluate these schemes does not provide much insight. Since obfuscating schemes do not completely eliminate leakage, the community widely agrees that these schemes do not achieve strong security properties, such as non-interference. Instead, it is valuable to ask if we can measure by *how much* they reduce leakage.

3.1 Limitations of Existing Work

To evaluate the effectiveness of an obfuscation scheme, we specifically desire a quantitative measure of an attacker's ability to distinguish between the victim's access patterns after they've been obfuscated. A quantitative metric measuring "distinguishability" should account for an optimal guessing strategy employed by the attacker. Such an optimal guessing strategy should guess the most likely victim access pattern by leveraging full knowledge of the obfuscating scheme's probabilistic properties.

Several previous attempts measure leakage by calculating the correlation between a victim's access trace and the attacker's observation trace (as in CSV [53] and SVF [10]). Such a measurement is not sufficient for measuring distinguishability, however. Even if an obfuscation scheme ensures that there is no statistical correlation whatsoever, an attacker can still learn information if it can distinguish between the victim's secret-dependent modulation patterns via its own observations.

Furthermore, an attacker's distinguishing ability is contingent on a large number of factors, depending on the microarchitectural obfuscation scheme, the attacker's methodology, and the victim's possible microarchitectural footprints. Prior work [2, 8, 10, 11, 14, 16, 20–22, 26, 27, 49, 53, 54] has failed to comprehensively model these factors, relying on simplifying assumptions such as only considering boolean victim secrets, passive attackers, or limiting their scope to persistent cache side-channels.

3.2 A Motivating Example

We now provide a motivating example to outline the quantitative goals of a desirable model for measuring obfuscation scheme leakage. In doing so, we will identify the questions that we expect a model to answer about a mitigation's leakage characteristics, and concretely demonstrate how to answer these questions.

Example Overview. We compare Prime+Probe attacks on four-line, fully-associative caches with different replacement policies. The attacker first primes the entirety of the four-line cache, waits for the victim to run, and then counts the number of cache misses it observes when re-accessing the four lines. The attacker's goal is to learn the number of distinct cache lines accessed by the victim during its execution.

As shown in prior work [29], the above attack is highly effective on a cache with a least recently used (LRU) replacement policy. By probing the cache's four lines in reversed order to which the lines were originally primed, the number of misses the attacker observes during the probe step is equal to the number of distinct victim cache line accesses (where four attacker misses represents *at least* four victim accesses).

Now consider a simple obfuscation mechanism that swaps the LRU replacement policy for a random replacement policy. The random replacement policy introduces non-determinism, obfuscating the characteristics of the victim's access patterns to the attacker. Figure 1 shows some of the possible cache states after the victim accesses a single cache line. When the victim accesses a single line, it randomly evicts one of the attacker's primed lines. If the victim evicts the attacker's first line, the attacker only observes one probe miss. If the victim evicts a different attacker line, however, the number of probe misses becomes non-deterministic. For instance, in the case where the victim evicts the fourth attacker line, the attacker may observe one miss (if accessing $_4$ evicts V_1), two misses (if accessing $_4$ evicts $_1$), and so on (up to four misses). Given this non-determinism, the attacker's ability to learn the number of victim accesses is degraded. As such, our security evaluation framework needs to measure by *how much* our attacker's ability degrades when employing such an obfuscation scheme.

Question 1: How can we quantitatively compare the effectiveness of obfuscation schemes? Does the random replacement policy lower the effectiveness of our Prime+Probe attack compared to using the LRU replacement policy?

We additionally identify several other important factors that can affect the security evaluation of an obfuscation mechanism. In the case of our random replacement cache, it is unclear if changing the attacker's number of probe addresses could improve the attacker's ability to learn about the victim's behaviour. A larger number of probing accesses may cause more victim collisions to be observable by the attacker. More attacker probe accesses, however, can result in *self-collisions*. A self-collision denotes the event wherein the attacker evicts its own line, making it more difficult to discern whether the victim evicted an attacker's line, or the attacker themselves. When examining an obfuscating defense mechanism it is thus also valuable to consider varying attack parameters, rather than assuming a single fixed strategy.

Question 2: How can we compare different attack strategies against an obfuscation scheme? How many lines should the attacker probe to *maximize* leakage?

Moreover, it is also possible for the victim application itself to impact the leakage of a mitigation scheme. Consider the side-channel leakage of a victim program with a boolean secret. Intuitively, distinguishing between the access patterns of a victim application which accesses 1 or 2 cache lines (depending on its secret) may be more difficult compared to distinguishing between the patterns of a victim application which accesses 1 or 4 cache lines. The reason is that in the latter case, the victim emits a stronger signal which can be easier to observe for the attacker. Furthermore, an application which has a wide variety of secrets (and corresponding access patterns) may leak more information than an application with a boolean secret. Thus the victim application itself should also be considered when assessing the leakage of an obfuscation scheme, as some victims' access patterns may be more susceptible to leakage under a certain obfuscation scheme than others.

Question 3: How can we quantify the leakage of varying victim applications under a given obfuscation scheme? Are there victim applications that issue access patterns which make them more vulnerable to a given attacker?

Concrete Quantification. We now present the analysis results enabled by Metior, answering the three questions posed for our random replacement cache example. We defer the explanation of *how* Metior computes the underlying leakage metric, and the mathematical meaning behind the metric, to the following sections.

Figure 2(a) compares the experimentally-obtained leakage of the LRU and random replacement schemes under varying attack strategies. In this figure, Metior measures the attacker's ability to distinguish between a specific victim's access patterns using its probe observations. The x-axis represents the number of cache lines probed by the attacker, and the y-axis represents the leakage of the attack. The victim application under study has a boolean secret,

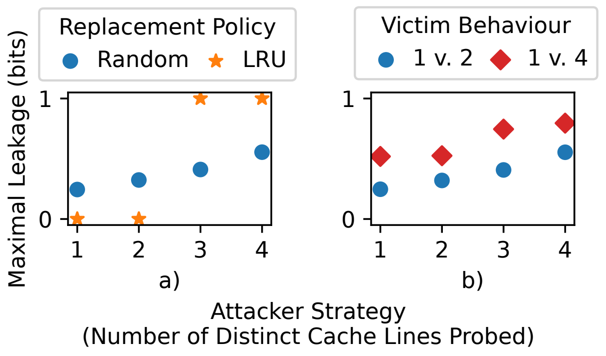

Figure 2: An example of the expected quantitative analysis result. Left: Leakage comparison of random replacement and LRU replacement policies across attack strategies. Right: Leakage comparison of different victims under a random replacement policy.

and accesses one cache line if its secret is 0, and two cache lines if its secret is 1.

As shown by the rightmost datapoints of Figure 2(a) (representing our original four-probe attack), LRU always leaks the victim's secret, while the attacker is only probabilistically successful under the random replacement scheme. Figure 2(a) further demonstrates the effectiveness of different attacker strategies against the random replacement cache. For the strategies considered in the plot, the attacker's leakage is maximized when probing four lines.

Figure 2(b) compares the leakage of differing victim programs when using our random replacement cache. In addition to the original victim program (which either issues 1 or 2 cache accesses), we also study a victim program which issues 1 or 4 cache accesses depending on the secret. Observe that for each attacker strategy studied, the leakage of the 1 or 4 victim is always higher than that of the 1 or 2 victim.

4 METIOR: A MODEL TO EVALUATE OBFUSCATING DEFENSE SCHEMES

In this section, we present Metior, a model to quantitatively evaluate obfuscating side-channel defense schemes which can be used to satisfy the three evaluation goals discussed above. Metior extends the scope of existing quantification work by considering wide ranges of microarchitectural side-channel defense schemes, addressing three key challenges to do so.

Challenge 1: Describing Obfuscated Information Flow. As discussed in Section 2.2, there exists many microarchitectural side-channel attacks, including those exploiting persistent and ephemeral channels, and those utilizing active and passive attack strategies. Note that microarchitectural events and interactions exploited by attackers can widely vary across different classes of attacks. For instance, a DRAM timing side-channel will manifest significantly differently compared to a spatial cache attack. Consequently, the obfuscating schemes that protect against different side-channel attacks can also differ substantially. This introduces our first research challenge: how to ubiquitously analyze a diverse range of obfuscating schemes under a unified model. We note that prior work has not addressed this research challenge before, as they

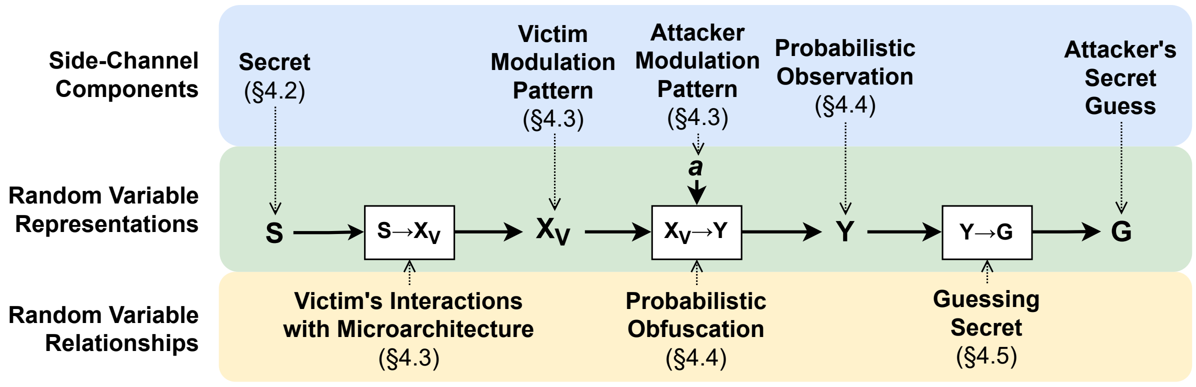
Side-Channel Components
Random Variable Representations
Random Variable Relationships
Secret (§4.2)
Victim Modulation Pattern (§4.3)
Attacker Modulation Pattern (§4.3)
Probabilistic Observation (§4.4)
Attacker's Secret Guess
S
S→X_V
X_V
a
X_V→Y
Y
Y→G
G
Victim's Interactions with Microarchitecture (§4.3)
Probabilistic Obfuscation (§4.4)
Guessing Secret (§4.5)

Figure 3: Metior Overview. For each side-channel component (top row), Metior maps the component to a random variable representation (middle row). Metior offers methodologies to derive the relationships (bottom row) between different random variables in order to derive a quantitative leakage metric.

either focus on a single type of channel [2, 6, 8, 11, 16, 20, 21, 54], such as caches, or only target passive attackers [8, 49].

Metior tackles this research challenge by describing the information flow through an obfuscating scheme using a discrete random variable representation, inspired by [49]. Figure 3 summarizes how we formulate each side-channel component (top row) as a random variable (middle row) to capture the end-to-end side-channel information flow. Each random variable represents a space of potential values. This random variable representation is extremely flexible and capable of considering wide classes of microarchitectural attacks and defense schemes.

Challenge 2: Defining Independent Modulation Spaces. Once we have mapped an obfuscation scheme to our random variable formulation, Metior attempts to quantify information leakage by deriving the probabilistic relationships between these random variables. For example, we aim to examine how the victim and attacker's modulation patterns interact inside an obfuscating scheme and lead to probabilistic observations by the attacker. This task is complicated by the fact that the victim's access patterns can be influenced by an active attacker's accesses (and vice-versa) due to contention in the shared microarchitectural structures they both access. This lack of independence complicates their representation, and poses difficulties in leakage derivation.

Metior addresses this challenge by representing attackers and victims using Directed Acyclic Request Graphs (rDAGs) [13]. In an rDAG, a node represents a request to use a microarchitectural resource, and an edge represents a dependency between the two connected requests with a fixed latency. We find that rDAGs can be used as a generic and independent representation to encapsulate modulation patterns for both attackers and victims. Using rDAGs enables us to use existing information theoretic metrics (which require independence between random variables) to derive leakage.

Challenge 3: Quantifying Leakage Under Non-Determinism. Our random variable formulation of obfuscated side-channels allows us to study the probabilistic/non-deterministic nature that often underpins obfuscating schemes. One challenge, however, lies in making the computations of these probabilities tractable, allowing the model to study real-world victim applications (e.g. AES and RSA) and state-of-the-art obfuscation mechanisms (e.g. CEASER-S and Camouflage).

Throughout this paper, we propose several practical solutions to address the engineering challenges associated with making these probability computations tractable. Using these derived probabilities, we leverage the Maximal Leakage metric, introduced in [49], to evaluate a scheme's leakage. The Maximal Leakage metric allows us to evaluate a defense scheme's *theoretical ability* to reduce the capability of a given attacker to distinguish between a victim's different behaviours.

4.1 Random Variable Representations

Metior's random variable characterization of side-channel information flow is summarized in Figure 3. The top row depicts a generic communication model for microarchitectural side-channels. It begins by considering a victim application with a secret. Depending on this secret, the victim accesses an obfuscating shared resource according to some modulation pattern. The attacker then issues its own modulation pattern to interfere with the victim's modulations of the channel, and observes the side effects of its modulations (e.g. timing). With its observations, the attacker then makes a guess of the secret.

At the core of Metior, we formulate each component in this side-channel model into a random variable as shown in the middle row in Figure 3. Each box in the middle row denotes the mapping relationship between random variables. The semantic meanings of these mapping relationships are listed in the bottom row.

In Metior, we denote the secret used by the victim as S, the victim modulation pattern as X_V, the attacker's modulation strategy as a, the attacker's side-channel observations as Y, and the attacker's guess as G. For each random variable, R, we represent its space of possible values as , and a singular value as r. For example, the victim's secret space is $\mathcal{S}$, and a concrete secret value is denoted as s. Note that discrete random variables are not limited to integer values. We can encapsulate a wide variety of representations (such as graphs or sets) within a random variable.

Secret S. The victim's secret S represents the information that the victim wishes to keep hidden from any attacker. In the case where S is a boolean secret, the set of possible secret values is $\mathcal{S}$ $\{0, 1\}$.

S can also denote *multi-bit* secrets. For instance, a cryptographic algorithm with a 256 bit key (such as AES) has a secret space of $\mathcal{S} \quad [0, 2^{256})$.

Victim's Modulation Pattern X_V. The victim's modulation pattern is the characterization of the victim program's access patterns to a particular shared microarchitectural structure. The victim's modulation pattern is a function of the victim program, its inputs (including the secret and public inputs), and the microarchitectural details of the machine the victim is running on.

Attacker's Modulation a. The attacker's modulation is the characterization of a fixed attacker's access patterns to the shared structure. The attacker's modulation can be broken down into two components: a preconditioning pattern a_{pre} and an observation pattern a_{obs}. a_{pre} aims to put the microarchitecture into a known state and is performed prior to the victim's accesses. a_{obs} aims to monitor the microarchitectural state via observable side effects when issued, either during or after the victim's accesses. For instance, for Prime+Probe, the access pattern to prime the cache is a_{pre}, and the pattern used to probe the cache is a_{obs}.

Attacker's Observation Y. The attacker's observation Y represents the (potentially non-deterministic) observation gathered via the attacker's observation operation (a_{obs}). The observation can have different representations depending on the attacker's modulation pattern. For instance, in a Prime+Probe attack, an observation y describes the number of misses the attacker encounters. In a memory controller timing side-channel, an observation y is the time to complete the attacker's requests. Since these observations can be non-deterministic, we note that the random variable Y encodes the probability distributions of observing different possible observations $y \in \mathcal{Y}$. One key task of Metior is to concretely derive these distributions.

Guess G. The attacker's guess of the secret is represented by G. The attacker's goal is to guess the victim's secret correctly, and thus to maximize the probability that $S \quad G$. Our security evaluation is thus tied to how successful an attacker is towards this goal.

Section Overview. With the above formulation, we now detail the key steps to apply Metior in the remainder of Section 4. Given an application, we first derive the victim's space of potential secrets (Section 4.2), then derive the possible victim modulation patterns for each of these secrets (Section 4.3). Next, given the mitigation scheme, the victim modulation pattern, and the attacker modulation patterns, we derive the attacker's non-deterministic observations in the form of conditional probabilities (Section 4.4). Finally, with these conditional probabilities in hand, we compute the amount of information leakage to measure the effectiveness of the obfuscating scheme under study (Section 4.5).

4.2 Step 1: Defining the Victim's Secret Space

To model a victim application's leakage, we first define the scope of the victim's *secret space*, which is used later to derive the victim's modulation patterns for each secret in this space. The victim's space of possible secrets is dictated by the semantics of the victim program, denoting the aspects of its execution that an attacker wants to extract via its observations.

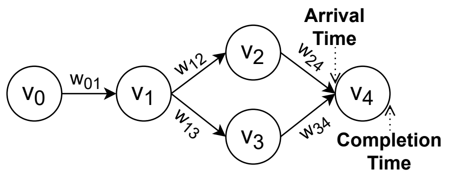

Figure 4: Example of a Directed Acyclic Request Graph (rDAG). Nodes represent requests (experiencing variable contention), edges represent request dependencies (with fixed latencies).

When analyzing the leakage of an encryption application, the evident secret space consists of the space of possible keys. Such a victim program may have several valid secret spaces however, each of which potentially having different leakage characteristics. For instance, an attacker targeting a cryptographic application may attempt to learn information about the victim's plaintext directly, rather than the encryption key.

Secret spaces can also be defined for non-cryptographic applications. Consider the analysis of a machine learning application. If the attacker attempts to learn the number of model layers, the secret space could be defined as $\mathcal{S} \quad [1, 16]$ layers. If the attacker instead attempts to learn the size of a given layer, the corresponding secret space might be $\mathcal{S} \quad \{16\times16, 32\times32, 64\times64\}$. Defining secret spaces requires domain knowledge of the application under study, and has been explored in other work [30, 39, 51].

4.3 Step 2: Defining the Modulation Spaces

After defining the victim's secret space, we now define the modulation patterns of the victim and attacker. To consider a concrete victim and attacker in our model, we first decide on a *representation* for their modulation patterns, describing these modulations in a symbolic way.

There are three desirable properties for a modulation representation. First, it should be general enough to encapsulate a modulation pattern's intrinsic features, characterizing both the spatial and temporal aspects of the attacker and victim's modulations. This allows Metior to model a wide variety of side-channels, both persistent and ephemeral. For instance, simply representing the victim's modulation pattern as a single number (as in our random replacement example) is insufficient to describe the victim's modulations when considering an ephemeral attack. Second, the modulation representation should be independent from the impact of any other applications sharing the microarchitectural structure. This allows Metior to study active attackers, avoiding the need to adjust victim modulation patterns for each attacker modulation pattern studied. Third, the modulation representation should be computationally feasible to derive, allowing for the derivation of the entire victim modulation space.

We find that there is no perfect modulation representation which exhibits all three of these properties. Metior can leverage *directed acyclic request graphs* ('rDAGs') to describe general and independent access patterns (properties 1 and 2), however this representation is often expensive to concretely compute (property 3). When this computational complexity is too high in practice, Metior can also leverage simpler representation patterns, such as access counts.

Table 1: Examples of how to choose modulation representations for different obfuscating defenses.

Obfuscating Defense Scheme	**Attack**	**Victim Modulation Space (X_V)**
Random Replacement (Set-Associative Cache)	Prime + Probe [29]	of Distinct Accesses (per set)
Random Replacement (Fully-Associative Cache)	Cache Occupancy [40]	of Distinct Accesses (total)
Randomized Cache [35, 48]	Probabilistic P+P [2]	of Distinct Accesses (per set)
Traffic Shaper [55]	Memory Contention [33]	Memory Access rDAG
Timer Degradation [31, 43]	Port Contention [1]	Port Access rDAG

rDAGs: General & Independent Representations. The rDAG notation used by Metior was first presented in DAGguise [13] for designing secure traffic shaping mechanisms. Using directed acyclic graphs to represent executions has also been explored in ZSim [37] for the simulation of CPUs.

An example rDAG is shown in Figure 4. Each node in an rDAG represents a request to a microarchitectural structure, while the edges between nodes represent the dependencies between these requests. Each directed edge has a weight corresponding to the time elapsed between the completion of the source node, and the dispatch of the destination node. For example, to describe modulation patterns to a memory controller, a node represents a memory request (a read or write) to a specific address in DRAM, and edges between nodes represent dependencies between these requests. The edge weights between nodes correspond only to time spent in the core and cache hierarchy, and are thus independent of contention in the controller.

rDAGs satisfy our first two desirable properties of a modulation pattern representation. First, they fully characterize the properties of modulation patterns leaked under both persistent and ephemeral side-channels, characterizing both the *spatial* and *timing* aspects of microarchitectural events. Second, rDAGs can describe modulation patterns in a conflict-agnostic way, ensuring that a modulation pattern is only a function of the program and its inputs, rather than a function of the mitigation scheme or any co-running applications. Such a representation allows Metior to reliably characterize both the attacker and victim's modulation patterns, representing them in a format which is independent from any activity in the channel.

Alternate Modulation Representations. While rDAGs offer a universal way to represent modulation patterns, from a computational perspective it is often desirable to use a simpler modulation representation. To calculate leakage, we will require knowledge of *all* possible victim modulation patterns for the given secret space. Note however that determining the entire space of possible victim rDAGs may be infeasible for large secret spaces.

In such cases, we can use alternative representations to represent victim modulation patterns. In order to avoid deteriorating the model's ability to consider the attacker's true abilities, however, the selected victim modulation representation should encapsulate all of the characteristics exploited by the studied attacker. In our random replacement cache example, the studied Prime+Probe attacker only leaked information about the number of distinct cache lines accessed by the victim. As such, the victim modulation pattern was simply represented as a count of distinct cache line accesses.

To further clarify the trade-offs in using different modulation representations, we show five examples in Table 1. Specifically, when modeling different obfuscating defenses on caches, we focus on modeling the spatial aspect of the modulation pattern and thus choose to use an integer-based representation to either count the number of distinct per-set accesses or total cache access as the modulation representation. When modeling obfuscating schemes such as traffic shapers and timer degradation, we focus on both the temporal and spatial aspects of the modulation pattern and thus choose to use rDAGs as the modulation representation.

Deriving Concrete Victim Modulation Patterns. Once the representation for the victim modulation patterns has been chosen, we then determine the concrete modulation pattern x_V for each secret in our derived secret space $s \in \mathcal{S}$. The exact requirements of this step are highly dependent on the microarchitectural structure and modulation representation chosen. We demonstrate several examples in the case studies, deriving cache access counts using static program analysis (Section 7), and rDAG representations for memory request patterns using a modified version of ZSim [37] (Section 9).

4.4 Step 3: Deriving Obfuscation Probabilities

Using the derived modulation patterns for the victim and attacker, we then derive the attacker's obfuscated observations. This process is denoted as the second box, $X_V \rightarrow Y$, in Figure 3. Given the non-deterministic nature of obfuscating schemes, we expect that the attacker's observations will exhibit complex probabilistic characteristics, which need to be precisely characterized for the correct measurement of side-channel leakage.

Recall that, due to obfuscation, for a given attacker and victim modulation pattern, there may be several possible observations. Thus, Metior describes the attacker's observation as a *probabilistic* random variable Y, denoting the conditional probability distributions of the attacker's observations for a given victim modulation pattern x_V and attacker modulation a, written as $\Pr[y|x_V, a]$.

For simple obfuscating schemes, it may be possible to construct closed-form expressions to derive the conditional probabilities $\Pr[y|x_V, a]$. When examining realistic obfuscation schemes, however, Metior estimates $\Pr[y|x_V, a]$ via Monte Carlo simulations. To do so, a simulator emulating the behaviour of the obfuscation scheme under study is required. For each attacker modulation a and victim modulation pattern x_V, we use such a simulator to observe the incidence rate (and thus the conditional probability) of each y.

Depending on the structure, deriving $\Pr[y|x_V, a]$ may require assuming an initial microarchitectural state. Metior generally assumes a known initial state (e.g. a cold cache), however, if desired, $\Pr[y|x_V, a]$ can be further augmented to probabilistically consider different initial microarchitectural states.

Passive attackers are a simpler sub-problem that can also be modeled by Metior. Since passive attackers do not perturbate the channel, Y is not dependent on any attacker modulation a. In these cases, the conditional probability distributions of the attacker's observations are simply defined as $\Pr[y|x_V]$.

It is usually desirable to understand the effectiveness of obfuscation schemes in a noise-free environment. Thus, in this paper, we do not consider noise, unless injecting noise into the protected system is considered as a scheme itself. In such a case, Metior's analysis flow can be adapted by deriving the attacker's noisy observations of the channel.

4.5 Step 4: Computing Leakage

As a final step, we use the derived conditional probabilities of the attacker's observations to evaluate the leakage of an obfuscation scheme. The goal of this leakage evaluation is to derive a consistent metric to compare the leakage of different victim applications, attack strategies, and obfuscation configurations. This process presents a maximization problem. Since different attacker modulation patterns may result in different amounts of leakage, it is desirable to search for an optimal attack modulation pattern in a specified attack space in order to maximize this leakage.

To quantify leakage we use the notion of *Maximal Leakage.* Maximal Leakage was first defined and studied by Issa et al. [23] and later used to specifically study side-channels by Wu et al. [49]. Intuitively, Maximal Leakage measures how much an obfuscated side-channel observation helps the attacker to make a correct guess of the victim's modulation pattern compared to a blind guess. Note that this metric provides a *relative* measurement, rather than an absolute number of bits being leaked.

More formally, Maximal Leakage is the upper bound of the multiplicative gain of the attacker's ability to guess the secret after an observation. The metric computes the multiplicative gain by maximizing it over any possible decoding strategy and comparing it to the success rate of an attacker taking a blind guess. Maximal Leakage has two equivalent definitions for an operational metric, one for intuitive reasoning and the other for computation.

Equation (1) is for intuitive reasoning, where we denote the attacker's observations when using a given attack modulation pattern a as Y_a : $\boxed{X \rightarrow Y}(a, X_V)$ and any possible decoding strategy as $\boxed{Y \rightarrow G}(\ )$.

$$\max_{\{S|(S \perp\!\!\!\perp Y_a|X\ \ )\}} \log_2 \left(\frac{\max_{\boxed{Y \rightarrow G}(\ )} \Pr[S \quad \boxed{Y \rightarrow G}(Y_a)]}{\max \quad \Pr[S \quad G]} \right) \tag{1}$$

The above expression of Maximal Leakage has been shown to be equal to the following directly computable expression (measured in *bits*). One of the nice features of the formula below is that it does not require knowledge of the distribution of X_V (see [23]).

$$L_{max}(X_V \rightarrow Y_a) \quad \log_2 \left(\sum_{y \in \mathcal{Y}} \max_{x \ \in} \Pr[y|x_V]\big] \right) \tag{2}$$

It is important to note that the above definition of Maximal Leakage only considers leakage from the victim's modulation space to the attacker's observation space. It does not say anything about the leakage from the victim's secret space.

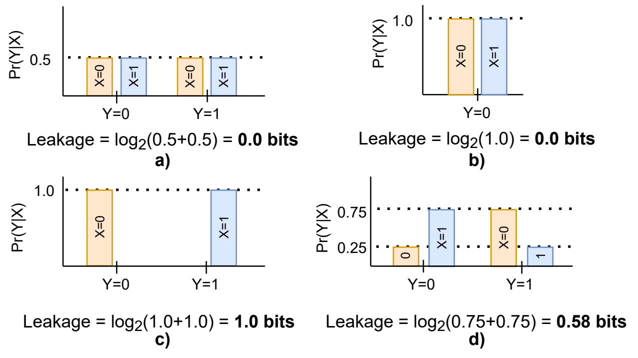

Figure 5: Example conditional probability distributions, and corresponding derived maximal leakage metrics.

Implications of Maximal Leakage. To examine the implications of Maximal Leakage, consider the four examples shown in Figure 5, which assume a victim modulation space of two elements (i.e., X 0 and X 1). In (a), the attacker's observations Y are equally likely, regardless of the victim's modulation pattern. Applying Equation (2), we observe that this results in 0 bits of leakage. This denotes that the attacker's ability to guess the victim modulation pattern is 2^0 1× (e.g. the same as) the probability of a blind guess, aligning with our intuition that no information can be leaked.

Example (b) also leaks nothing, since there is only one possible observation for the attacker, with this observation being identical across the two different victim modulations.

In (c), there is a one-to-one correspondence between the victim's modulation pattern and the attacker's observation. Maximal Leakage denotes 1 bit of leakage in this case. Note that this 1 bit of leakage does not indicate the leakage of 1 bit of the secret. Instead, this indicates the attacker can guess the correct victim pattern 2^1 2× better than a blind guess. Since a blind guess has a 50% probability of being correct, 2× of it means 100% accuracy, meaning a complete leakage of the victim's modulation pattern.

In (d), the attacker only probabilistically learns about the victim's modulation pattern. For example, when X 0, there is a 25% chance that the attacker observes Y 0, and a 75% chance that the attacker observes Y 1. As a result, Maximal Leakage computes less than 1 bit of leakage.

Maximal Leakage should be used to compare different obfuscation scenarios with caution, since context plays a large role in its interpretation. When distinguishing between two modulation patterns, a leakage derivation of 1.0 bit corresponds to complete disclosure of the victim's access patterns (e.g. Figure 5(c)). Conversely, when attempting to distinguish between 64 victim modulation patterns, leakage of 1 bit denotes a marginal increase in distinguishing ability (i.e. from a $\frac{1}{64}$ blind guess to a 2^1 $\frac{1}{64}$ $\frac{1}{32}$ chance). Metior's Maximal Leakage derivations should *not* be used as an absolute vulnerability metric to compare different defenses (e.g. comparing a persistent cache defense vs. an ephemeral memory controller defense), and should only be directly compared if the victim's modulation spaces are equivalently sized.

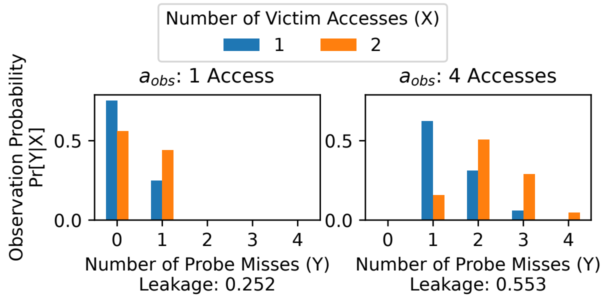

Figure 6: Derived conditional probability distributions of attacker observations on a random replacement cache. Left: Attacker probing one line. Right: Attacker probing four lines.

We note that Metior can employ other information theoretic leakage metrics if desired, such as local differential privacy [23] or mutual information [55]. Maximal Leakage has practical advantages over these metrics by providing an intuitive measure of leakage which can be tractably estimated via sampling. A detailed comparison of these metrics can be found in [49].

4.6 Revisiting Random Replacement

We now revisit the random replacement example outlined in Section 3.2 to illustrate how we can apply Metior to study end-to-end obfuscation leakage. Recall that we studied a victim that has a one-bit secret S, which maps to a victim modulation of either 1 or 2 cache accesses ($x_V \quad s+1$, $\mathcal{X}_V \quad \{1,2\}$). The attacker's modulation pattern a_{obs} describes the number of cache accesses the attacker performs during the probing step, from 1 to 4. For each attacker and victim modulation, we use a Monte-Carlo simulation to find the observation probabilities (running the simulation for 20,000 rounds to allow for the probabilities to converge).

In Figure 6 we show the derived conditional probabilities for two attacker modulation patterns ($a_{obs} \quad \{1,4\}$). We then derive the Maximal Leakage by applying Equation 2 to these computed probabilities. This computation results in Figure 2. Observe that the leakage induced by the 4 probe attacker is significantly higher compared to the 1 probe attacker (0.553 vs. 0.252 bits). While the probabilities in Figure 6 were derived via Monte Carlo simulations, we note that calculating the leakage using probabilities derived from an analytical model of the cache (i.e. the true conditional probabilities) results in nearly identical Maximal Leakage results (0.562 and 0.247 bits, respectively).

5 LIMITATIONS AND CLARIFICATIONS

We note several limitations when using Metior in practice, which we classify into two categories: Metior's computational complexity and the interpretation of its leakage values.

Computational Complexity. When employing Metior to study real-world applications and defense schemes, we acknowledge that challenges can arise when deriving large victim modulation spaces $\mathcal{X}_V$, or when computing the closed-form conditional probability expressions $\Pr[y|x_V, a]$.

When a victim program has a huge secret space, the space of possible victim modulation patterns $\mathcal{X}_V$ may be correspondingly large. For instance, deriving the rDAGs for each of the possible 2^{256} secret keys in a cryptographic algorithm's secret space is infeasible. This can sometimes be alleviated by using an alternative modulation representation that solely encapsulates the information used by the attacker, reducing the modulation space size. For instance, in Section 7 we showcase an example where symbolic execution is leveraged to derive an alternative representation for AES when studying cache occupancy attackers. By applying alternative representations and studying victim programs with smaller secret spaces, Metior can already offer valuable insights into the effectiveness of state-of-the-art obfuscation mechanisms.

Additionally, we do not expect to be able to analytically derive the closed-form expression $\Pr[y|x_V, a]$ for nontrivial obfuscating schemes. As such, Metior relies on Monte Carlo simulations to derive conditional probabilities. Since Monte Carlo simulations can yield some degree of uncertainty, we take care to ensure that the derived probabilities converge before calculating leakage.

Interpretation of Leakage Value. The Maximal Leakage value generated by Metior should be interpreted as a *relative* metric to compare an attacker's ability to distinguish between a victim's obfuscated modulation patterns. Metior can be used to directly compare the leakage across different attacker strategy spaces (demonstrated in Section 8). It cannot, however, reason about attacks outside of an examined space, nor can it generate novel attack classes. Despite this, we find that studying existing attack strategies on complex obfuscation mechanisms already leads to new insights.

When interpreting the leakage value of a victim application, one should note that the semantic meanings of "the secret being leaked" and "the modulation pattern being distinguished" are two related, but different things. For instance, determining the victim's modulation pattern may result in the direct leakage of a secret bit (such as in RSA), or may leak some second-order information (e.g., narrow down the set of possible AES keys). By decoupling from the semantics of the studied program, only measuring the level of distinguishability of obfuscated modulation patterns, Metior can focus on the leakage from the obfuscation defense scheme itself.

6 CASE STUDY OVERVIEW

To demonstrate Metior's ability to study obfuscating defenses, we use the remainder of this paper to provide three case studies showcasing Metior. Throughout these case studies, we focus on discussing how various engineering challenges can be addressed when using Metior, including how to map real programs to concrete secret and modulation spaces which make leakage computations tractable, and how to concretely derive observation probabilities via simulation.

We provide a preview of the three case studies below. For each case study we describe the concrete questions that drove us to design it, and summarize new and insightful findings that we think could benefit the community.

In Case Study I (Section 7), we aim to offer a case study that exhibits many of the possible engineering challenges in the evaluation of obfuscating schemes. In doing so, we try to answer the following two questions. First, is it feasible to study a victim program with a large secret space? Second, can Metior indeed be used to quantitatively compare different attack strategies and identify an optimal strategy that maximizes leakage? To this end, we choose to study the effectiveness of random replacement caches in obfuscating an AES victim's memory access patterns against a cache occupancy [40] attacker.

In Case Study II (Section 8), we extend our security analysis to compare different classes of attacks. Specifically, we study a state-of-the-art randomly mapped cache, Skewed-CEASER [35], comparing the leakage of cache occupancy attacks and Probabilistic Prime+Probe (PPP) attacks [34]. We note that the community has considered these two attacks to be distinct, and has often separated them into distinct threat models [36]. Our evaluation results offer new insights to help understand the working mechanisms behind PPP, casting doubt onto this claim. We find that, under certain obfuscating configurations, PPP works by primarily exploiting cache occupancy effects, rather than exploiting targeted conflicts.

Finally, in Case Study III (Section 9), we are driven to show whether Metior can be used to evaluate obfuscating schemes that mitigate ephemeral side-channels. As there exists obfuscating schemes that involve tunable parameters to trade-off between security and performance, we are additionally curious to see whether Metior can explore this trade-off space. With the above pursuit in mind, we choose to evaluate Camouflage [55], a DRAM traffic shaper which shapes a victim's traffic patterns to a tunable timing distribution. The presumption of such a tunable scheme is that by slightly relaxing security, we expect to see some small performance benefits with a marginal amount of security loss. However, Metior reveals that Camouflage has a complex security/performance trade-off space, identifying scenarios where a significant amount of information can be leaked even with a small amount of relaxation.

7 CASE STUDY I: COMPARING ATTACK STRATEGIES

We first set out to provide an end-to-end demonstration of Metior in studying the effectiveness of randomly mapped caches in protecting the S-Box implementation of AES [44], an application with a large secret space. To this end, we choose to study the side-channel leakage of fully-associative caches which employ *random replacement* policies. The use of fully-associative random replacement schemes has been a desirable choice to attain security since they do not allow for fine-grained set-conflict attacks. While fully-associative random replacement schemes (and schemes which emulate their properties, such as MIRAGE [36]) block set-conflict attacks, their lack of strict partitioning still leaves them vulnerable to cache occupancy attacks [40]. As noted in Section 3.2, a cache's replacement policy plays a large role in the amount of leakage observed in a fully-associative cache. We are curious to know by how much random replacement can help mitigate cache occupancy attacks which target real-world victims with small memory footprints.

Defining the Victim's Spaces $\mathcal{S}$ and $\mathcal{X}_V$. We examine an attacker who is trying to learn about the victim's secret key, thus the secret space examined is $\mathcal{S}$ $[0, 2^{256})$. An occupancy attacker attempts to distinguish the number of distinct cache lines accessed by the victim (a function of the secret key), thus x_V can be represented by an integer access count (avoiding the complexity of rDAGs).

Since the secret space is very large, it is infeasible to directly derive the access counts for each possible secret key via brute-force simulation. Instead, to determine the set of possible access counts

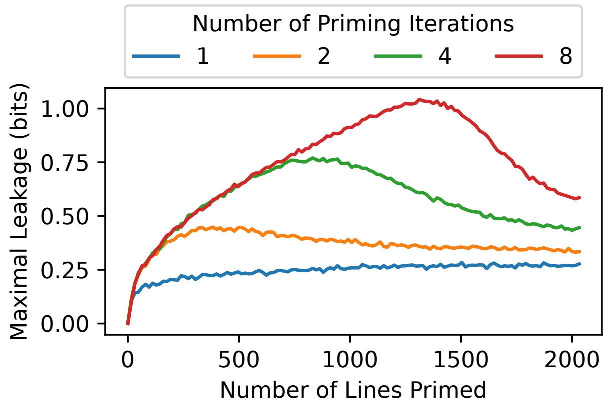

Figure 7: Leakage across varying attacker strategies when distinguishing between AES access patterns. Increasing priming iterations increases leakage, but priming too many lines can result in a leakage *decre se* due to self-conflicts.

$\mathcal{X}_V$ we use KLEE [5], a program-level symbolic execution framework. KLEE derives the relationship between a symbolic input (the secret key) and the number of distinct cache line accesses the victim performs (assuming 64B cache lines). For every possible cache access count (bounded between zero and the maximum number of memory accesses issued by the program), KLEE uses a SMT solver to determine if it is possible for the victim to issue a given count. If a concrete key is found which results in a given distinct cache line count, that count is included in $\mathcal{X}_V$. Using our KLEE workflow to study the AES program under evaluation, we determine that between 15 and 26 distinct cache lines (inclusive) are accessed in the first round of encryption.

Attacker Modulation a and Observation Space $\mathcal{Y}$. A cache occupancy attacker's modulation a consists of two steps, a_{pre} and a_{obs}. Using a_{pre}, the attacker primes (pre-conditions) the cache by sequentially accessing a set of addresses (the *priming set*). As the attacker does not target specific cache sets, these addresses (with distinct tags) are chosen arbitrarily. The addresses can be accessed multiple times to increase the probability of installing all lines before the victim runs. After the victim runs, the attacker probes the cache by re-accessing the primed addresses (i.e. a_{obs}). The attacker's observation y denotes the number of misses during this probing step.

Comparing Leakage of Varying Attack Parameters. We compare the leakage of a space of cache occupancy attacks, varying two parameters. First, we vary the number of elements in the attacker's priming set, increasing the number of cache lines monitored. Second, we vary the number of times the attacker iterates over the priming set, raising the likelihood that more of the priming set is installed before probing.

We simulate a 128kB fully-associative cache with random replacement. We estimate the conditional probability distributions by simulating the outlined attack 10,000 times for each victim modulation pattern x_V and attacker modulation pattern a, then estimate the values of $\Pr[y|x_V, a]$.

The Maximal Leakage results derived by Metior are shown in Figure 7, highlighting key insights about different attacker strategies.

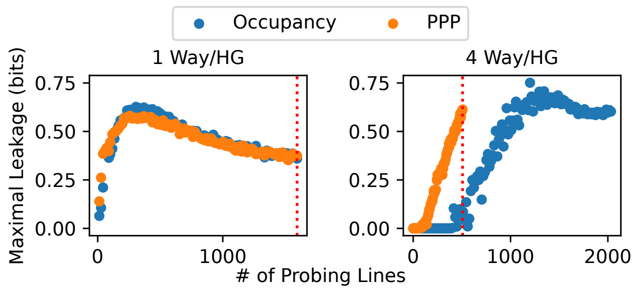

Figure 8: Leakage comparison between Probabilistic Prime+Probe and occupancy attacks for different Skewed-CEASER configurations.

Across all attackers considered, Metior denotes an optimized leakage of 1.03 bits when priming 1319 out of 2048 total lines, with 8 priming iterations. Recall that this is a multiplicative leakage, representing an upper-bound on the theoretical gain of distinguishability compared to a blind guess. That is, the 1.03 bit attacker is about twice ($2^{1.03}$) as successful than a blind guess in guessing the victim access pattern. Again, we emphasize that this leakage derivation does not equate to leaking 1.03 bits of the secret key directly.

Key Takeaways. By sweeping attacker parameters, Metior offers several key insights into the underlying leakage behaviour of fully-associative random replacement caches during occupancy attacks. Increasing the number of priming iterations monotonically increases leakage, as more priming iterations increases the expected number of primed lines resident in the cache. Increasing the size of the priming set, however, sometimes *reduces* leakage. Since increasing the size of the priming set increases the number of self-conflicts, it is more difficult for an attacker to distinguish conflicts caused by the victim, and conflicts caused by its own accesses.

8 CASE STUDY II: COMPARING ATTACK CLASSES

In the second case study, we showcase the use of Metior in the quantitative comparison of different defense scheme configurations, and the vulnerability of these configurations against different classes of attacks. We choose to analyze the leakage of Skewed-CEASER (CEASER-S) [35], a popular randomly mapped cache defense scheme. We compare the effectiveness of two classes of attacks against CEASER-S: Probabilistic Prime + Probe [34] (PPP) and cache occupancy. In doing so, we gain a better understanding of how PPP attacks leak information, revealing new insights that prior work [2] did not offer.

Evaluation Setup. In line with prior work [2] that evaluates the security property of CEASER-S against PPP attacks, we examine attackers who attempt to leak a single key bit of an RSA victim ($\mathcal{S}$ $\{0, 1\}$). The victim accesses 10 distinct lines if the secret key bit is 0, and 26 distinct lines otherwise (thus, $\mathcal{X}_v = \{10, 26\}$). We examine two 16-way 128KB cache configurations of CEASER-S: one configuration using 1 way per hash group (HG), and the other using 4 ways per HG (with an LRU policy inside each HG).

Comparing PPP and Occupancy Attacks. In Figure 8, we show the Maximal Leakage of PPP and occupancy attacks when varying the number of probing lines used in each attack. In a PPP attack, the attacker constructs an eviction set using an eviction set generation strategy [34, 35] to find addresses that map to the same sets as the victim's accesses. This step is referred to as the *calibration step*. With the eviction set, PPP can only probabilistically collide with the victim's addresses. Prior work [2, 34] considers the calibration step to be key in the success of PPP attacks.

In line with CaSA [2], we configure the PPP attacker to use 1024 candidate addresses and repeat the calibration steps up to 200 times to ensure calibration can be completed within the scheme's re-keying window. The maximum number of probing lines that PPP can generate within this interval is denoted using the vertical red dotted line. The attacker uses the eviction set generated during calibration to Prime+Probe the cache, aiming to observe targeted collisions with the victim.

We observe that the leakage characteristics for the attacks targeting the 1 way/HG Skewed-CEASER configuration are practically indistinguishable under the PPP and occupancy attacks studied. This indicates that the occupancy effects dominate the attacker's ability to distinguish between the victim's modulation patterns, with little to no leakage improvement gained from calibration. This new observation suggests that a PPP attack on a 1 way/HG CEASER-S configuration may primarily exploit leakage via occupancy.

In the 4 way/HG case, when probing the cache with an equivalent number of addresses, the leakage of PPP is significantly higher than an occupancy attack, indicating that calibration is indeed successful in improving a PPP attack's ability to distinguish between the victim's modulation patterns. Observe however that occupancy attacks leak a roughly equivalent amount when a larger number of probing addresses are used.

Key Takeaways. By quantifying and comparing the leakage of different attack strategies, Metior can help to identify the underlying mechanisms which make an attack strategy successful. Prior work [2] presumed that Probabilistic Prime+Probe (PPP) attacks relied on targeted collisions with the victim's accesses to leak information. This led to major efforts towards constructing eviction sets. Based on our analysis, however, we find that the attack primarily exploits occupancy effects under certain obfuscation configurations (e.g. 1 way/HG CEASER-S). In these cases, extra eviction set construction efforts do not help increase leakage. This observation suggests that when designing obfuscating schemes for certain real-world victim applications, we should take occupancy attacks into consideration, as they can sometimes achieve a similar level of leakage as set-conflict attacks.

9 CASE STUDY III: STUDYING EPHEMERAL OBFUSCATION SCHEMES

In the last case study, we demonstrate Metior's use in analyzing the leakage of Camouflage [55], an obfuscation scheme which targets ephemeral channels. Camouflage defends against memory timing side-channels by shaping the victim's traffic patterns to a tunable, secret-independent distribution. While the selection of this distribution provides a trade-off between security and performance, no prior work has studied the security implications of Camouflage's different defense configurations.

Camouflage Overview. Camouflage obfuscates the timing of a victim's DRAM requests by shaping their inter-request timing to

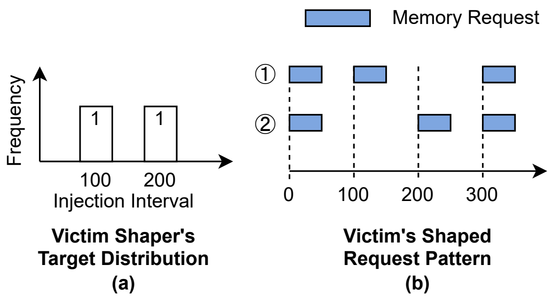

Figure 9: An example of Camouflage's inability to hide fine-grained request patterns.

a secret-independent distribution, delaying the victim's requests and issuing fake requests when necessary. The shaping distributions are represented using a histogram, with each histogram bin containing a number of credits. Each histogram bin represents a certain inter-arrival time interval, and each credit represents one memory transaction that can be issued to fulfill that interval.

We evaluate an implementation of Camouflage which shapes the memory requests of the victim. The shaper enforces the desired timing distribution by delaying a memory transaction if there are no credits available in the bin representing the request's current inter-arrival time. Requests which are issued between bin intervals are delayed until the next bin, and a fake request is always issued if the current inter-arrival time matches the last remaining bin with credits (ensuring that the shaping histogram is always adhered to). Bin credits are replenished to recover the shaping distribution after all have been consumed.

Camouflage's Ordering Vulnerability. As discussed in the Camouflage paper [55] (and further in [13]), Camouflage still leaks some information through the fine-grained ordering of the selected memory request intervals, despite faithfully shaping traffic to a secret-independent distribution. Since the *ordering* of the timing intervals drawn from the distribution is not fixed by Camouflage, the output of the shaper is not independent from the victim's secret-dependent traffic. Thus, two different victim access patterns may be shaped to two different shaped access patterns. Consider the shaping example shown in Figure 9, wherein the shaping distribution (shown in Figure 9a) allows for one 100-cycle interval and one 200-cycle interval. Depending on the underlying victim modulation pattern to be shaped, the shaper may generate one of two access patterns (Figure 9b ① or ②). An attacker who can distinguish between these two shaped access patterns can thus learn information about the victim's original modulation pattern (i.e. the DRAM access pattern being obfuscated by the shaper).

Camouflage's existing security evaluation derives the mutual information between the overall traffic distributions before and after the shaper, ignoring the ordering effect. As such, this existing evaluation only provides high-level insights into the leakage of attackers who cannot observe fine-grained timing information. We are interested in using Metior to improve upon and extend this analysis, studying the leakage of attackers which leak information via the fine-grained latencies of their own accesses.

Evaluation Setup. We study the leakage of nine different shaping histograms, each containing 20 credits distributed over five bins,

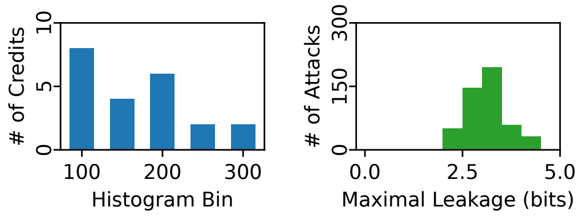

Figure 10: Left: (8,4,6,2,2) shaping configuration. Right: Leakage for varying attack strategies under (8,4,6,2,2) config.

$$\begin{tikzcd} A_0 \arrow[r, "50{,}000 / N"] & A_1 \arrow[r] & \cdots \arrow[r] & A_{N-1} \arrow[r, "50{,}000 / N"] & A_N \end{tikzcd}$$

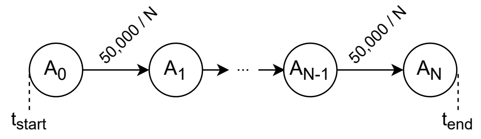

Figure 11: Attacker Modulation Pattern Template

which are selected to represent the wide variety of shaping distributions available. Each histogram configuration is defined by a 5-tuple, where each element of the tuple is the number of credits for a bin representing an inter-arrival time of 100, 150, 200, 250, or 300 cycles. Figure 10(a) shows the histogram configuration $(8, 4, 6, 2, 2)$, representing a timing distribution obtained via profiling a representative execution. Alternatively, the $(20, 0, 0, 0, 0)$ histogram represents a configuration where Camouflage issues requests at a constant rate of 100 cycles.

We study the ability of Camouflage to protect DocDist [19], a program calculating the euclidian distance between words in a public and private document. We measure the attacker's ability to discern between the DRAM access patterns of 64 different private documents ($\mathcal{S}$ $[0, 64)$). For each of the 64 documents, we generate a memory access pattern rDAG representing the first fifty-thousand instructions of the secret-dependent region using a modified version of ZSim [37]. The attacker attempts to discern which modulation pattern (rDAG) the victim issued, thus the highest leakage an attacker could possibly observe is 6 bits (representing a 2^6 $64\times$ multiplicative improvement of guessing correctly after its observation, compared to a blind guess). To derive observation probabilities, we model a memory controller employing Camouflage using a first-come first-serve scheduling policy with an unpipelined access latency of 100 cycles.

We examine varying ephemeral attackers which issue a series of dependent DRAM accesses. We consider the space of attacker rDAGs representing modulation patterns containing between 2 and 500 memory requests, with uniform edge weights summing to 50,000 cycles. This space of considered attacker rDAGs is shown in Figure 11. The attacker's observation y is the total number of cycles needed to complete its accesses.

Results. A summary of the observed leakage for all nine shaping distributions considered is shown in Figure 12. We report the mean and the maximum value of the Maximal Leakage observed when sweeping across the entire space of considered attacker modulation patterns. When no shaping is employed (denoted by the horizontal lines), an optimized attacker leaks all six bits of information (5.93 bits on average). When shaping is employed, the optimized Maximal Leakage of all shaping histograms considered exceeds 4 bits,

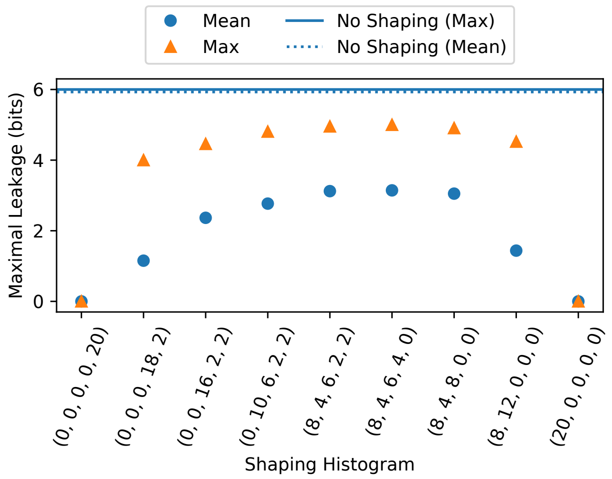

Figure 12: Leakage of Camouflage employing different shaping distributions, considered over differing attack strategies.

indicating significant amounts of leakage across all (non-constant-rate) shaping distributions considered. Note that while single-bin configurations guarantee no leakage, they are undesirable since they do not consider the victim's dynamic memory requirements (potentially resulting in poor performance). A detailed breakdown of the leakage observed for a selected histogram (8,4,6,2,2) across all attacker strategies considered is shown in Figure 10(b).

Intuitively, one might expect that by increasing the flexibility of the shaping distribution (improving performance), Camouflage's leakage will gradually increase in tandem. However, we observe that such an intuition can sometimes be incorrect. Considering the highest observed leakage across all attacker strategies, the single-bin distributions leak 0 bits, while a slightly relaxed distribution (e.g. (0,0,0,18,2)) leaks 4.5 bits about the victim's modulation pattern. We observe relatively similar maximized leakage across the remaining non-constant-rate distributions considered.

Key Takeaways. The above analysis demonstrates that the trade-off space between security and performance in a tunable obfuscation scheme can be non-linear. In the case of Camouflage, one may presume that slightly relaxing the constraint of the traffic shaper will buy the user a small amount of performance benefit with a small amount of security loss. However, our analysis demonstrates that even when the shaping distribution is slightly relaxed from the constant-rate case, significant amounts of information about the victim's modulations could still be leaked. This demonstrates that great care should be taken when employing Camouflage, since significant amounts of information about the victim's modulations could still be leaked by a side-channel attacker – even when the shaping distribution is slightly relaxed from the constant-rate case.

10 RELATED WORK

We now compare Metior with prior work on side-channel analysis, focusing on the modeling of the victim program space, the attackers considered, and their leakage derivations.

Modeling Victim Space. Very little work comprehensively considers the victim's secret space and modulation patterns. CSV [53] and SVF [10] do not have a notion of victim spaces. Hunger et al. [22] focus on analyzing covert channels communicating a boolean secret. CacheFx [20] evaluates randomly-mapped caches assuming a boolean secret. This overly simplified modeling is also used by He et al [21]. CaSA [2] and Domnitser et al. [14] consider more diverse cache access patterns, but still in boolean secret context. Köpf and Basin [26, 27] present a model studying the direct leakage of a secret, however their model can only be applied to study *deterministic* systems/side-channels.

Modeling Different Attack Classes. Prior work on modeling the attacker space is also far from comprehensive. Wu et al. [49] and CHALICE [8] can only target passive (rather than active) attacks. A line of work exists analyzing cache side-channel attacks [2, 6, 8, 11, 16, 20, 21, 54]. These approaches are limited to studying persistent side-channels, and cannot be applied to ephemeral ones. CacheAudit [16] over-approximates a program's possible cache states via static analysis to derive conservative leakage estimates – we note that deriving the possible states for randomized caches (a large space) will not lead to useful bounds.

Leakage Metric. Metior uses Maximal Leakage [49] as its metric, which offers a tractable and interpretable leakage value. Hunger et al. [22] uses mutual information to analyze covert channels. CSV [53] and SVF [10] uses the correlation between the victim and the attacker's traces, however these metrics are not useful in analyzing obfuscation techniques which distort the victim's modulations. CaSA [2], He et al. [21], and Domnitser et al. [14] use the probabilities of observing certain cache events as a leakage metric assuming a boolean secret space.

11 CONCLUSION

We presented Metior, a comprehensive random variable model that evaluates the effectiveness of obfuscating side-channel defense schemes. Metior offers key contributions in describing the side-channel information flow through these schemes for wide classes of attacks, including those which leverage both persistent and ephemeral side-channels. By extending existing work from information theory to quantify this flow, we have shown that Metior reveals interesting leakage behaviours of state-of-the-art obfuscating schemes. We hope to further inspire the community to leverage Metior to explore how different defense parameters, victims, and attacker classes affect the side-channel leakage of new and existing obfuscation schemes.

ACKNOWLEDGMENTS

This work was funded in part by the NSF under grants CNS-2046359 and CCF-2217099, and the Air Force Office of Scientific Research under grants FA9550-20-1-0402 and FA9550-22-1-0511. Additional funding was provided via a gift from Intel, and an MIT RSC research fund. We thank the anonymous reviewers and our shepherd for all of their valuable feedback during the review process.

A ARTIFACT APPENDIX

A.1 Abstract

Our artifact is comprised of a cache model, an occupancy attack implementation, and a Maximal Leakage calculator. The cache model can be configured to simulate various different cache architectures (including fully-associative random replacement caches and CEASER-S) under varying configurations. The occupancy attack implementation leverages this cache model to simulate the attacker's (probabilistic) observations for a given attacker/victim scenario. The top-level script (`runSweep.py`) samples a large number of observations from this implementation, sweeping across varying attacker strategies for a given victim space. Using these samples, our artifact provides an estimate of an attack's Maximal Leakage.

We have provided an end-to-end flow to replicate the results of Case Study I/Figure 7, demonstrating the leakage of cache occupancy attacks waged against an AES implementation for a globally random-replacement cache architecture. The provided scripts sweep varying cache occupancy attacker strategies (including the number of lines primed, and the number of priming iterations). This flow can be extended by researchers to easily derive the Maximal Leakage values for their own simulations.

A.2 Artifact check-list (meta-information)

- **Algorithm:** Maximal Leakage derivation via Monte Carlo simulation.
- **Run-time environment:** Python 3.
- **Output:** Maximal Leakage value (in bits) for varying attacker configurations, presented in a format identical to Figure 7.
- **Experiments:** Cache occupancy attack attempting to leak AES victim modulation patterns.
- **How much disk space required (approximately)?:** 100KB
- **How much time is needed to prepare workflow (approximately)?:** 5 minutes
- **How much time is needed to complete experiments (approximately)?:** 24-48 Hours (on 96 core machine).
- **Publicly available?:** Yes, available at https://github.com/CSAIL-Arch-Sec/Metior
- **Code licenses (if publicly available)?:** MIT License.
- **Archived (provide DOI)?:** 10.5281/zenodo.7823897.

A.3 Description

A 3 1 How to access Our cache models, simulator, and Maximal Leakage calculator is available at https://github.com/CSAIL-Arch-Sec/Metior.

A.4 Installation

Our infrastructure is built using Python 3. We require several Python dependencies for execution and figure plotting, which can be installed by running:

```
pip3 install  r requirements.txt
```

A.5 Experiment workflow

The entire experimental workflow is managed by the top-level `runSweep.py` script, which is run by calling:

```
python3 runSweep.py
```

The 'Experiment Parameters' section of this script defines the attacker strategies and victim access patterns to be examined. For each unique attacker strategy and victim access pattern, an instance of `singleSub` (in `runSweep.py`) will run on a single core. Thus, varying attacker strategies will simultaneously run on all cores on the experiment machine. A single instance of `singleSub` will examine the leakage of the victim for its assigned attacker strategy, performing a Monte Carlo simulations of an occupancy attack. Once the Monte Carlo simulations have completed, the leakage value for the studied attacker/victim pair is returned by `singleSub`.

Once the simulations for *all* attacker strategies have completed, the Maximal Leakage results are saved to an archive file (`results.pkl`). The results can be visualized (as in Figure 7) by running:

```
python3 plotResults.py results.pkl
```

A.6 Evaluation and expected results

The results generated/plotted from this artifact should match those shown in Figure 7. Some negligible statistical noise may be observed (due to the probabilistic sampling nature of Metior), however they should not impact the overall trends and takeaways observed.

A.7 Experiment customization

If desired, additional attacker strategies can be examined by augmenting the 'Experiment Parameters' section of `runSweep.py`.

It is important to note that the simulation time is heavily influenced by the number of Monte Carlo iterations performed by the cache occupancy algorithm. Figure 7 was derived by running the algorithm with 20,000 iterations (`cache.iterations = 20000`), however this takes several days on a 96 core server. Since this may not be desirable for rapid evaluation, we suggest running the script with far fewer iterations (e.g. `cache.iterations = 1000`), which will result in a noisier plot, but with the same overall trends.

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

REFERENCES

[1] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. 2019. Port Contention for Fun and Profit. In *2019 IEEE Symposium on Security and Privacy SP).* IEEE. https://doi.org/10.1109/SP.2019.00066

[2] Thomas Bourgeat, Jules Drean, Yuheng Yang, Lillian Tsai, Joel Emer, and Mengjia Yan. 2020. CaSA: End-to-end Quantitative Security Analysis of Randomly Mapped Caches. In *Proceedings of the 53th Annual IEEE/ACM International Symposium on Microarchitecture MICRO).* IEEE. https://doi.org/10.1109/MICRO50266.2020.00092

[3] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Sizhuo Zhang, Arvind, and Srinivas Devadas. 2019. MI6: Secure Enclaves in a Speculative Out-of-Order Processor. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *MICRO '52).* Association for Computing Machinery, New York, NY, USA, 42–56. https://doi.org/10.1145/3352460.3358310

[4] M. Busi, J. Noorman, J. Bulck, L. Galletta, P. Degano, J. Muhlberg, and F. Piessens. 2020. Provably Secure Isolation for Interruptible Enclaved Execution on Small Microprocessors. In *2020 IEEE 33rd Computer Security Foundations Symposium CSF).* IEEE Computer Society, Los Alamitos, CA, USA, 262–276. https://doi.org/10.1109/CSF49147.2020.00026

[5] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.. In *OSDI*, Vol. 8. 209–224.

[6] Pablo Cañones, Boris Köpf, and Jan Reineke. 2017. Security Analysis of Cache Replacement Policies. *CoRR* abs/1701.06481 (2017). arXiv:1701.06481 http://arxiv.org/abs/1701.06481

[7] Sunjay Cauligi, Gary Soeller, Fraser Brown, Brian Johannesmeyer, Yunlu Huang, Ranjit Jhala, and Deian Stefan. 2017. FaCT: A Flexible, Constant-Time Programming Language. In *2017 IEEE Cybersecurity Development SecDev)*. 69–76. https://doi.org/10.1109/SecDev.2017.24

[8] Sudipta Chattopadhyay, Moritz Beck, Ahmed Rezine, and Andreas Zeller. 2019. Quantifying the information leakage in cache attacks via symbolic execution. *ACM Transactions on Embedded Computing Systems TECS)* 18, 1 (2019), 1–27.

[9] Miles Dai, Riccardo Paccagnella, Miguel Gomez-Garcia, John McCalpin, and Mengjia Yan. 2022. Don't Mesh Around: Side-Channel Attacks and Mitigations on Mesh Interconnects. In *31st USENIX Security Symposium USENIX Security 22)*. USENIX Association, Boston, MA, 2857–2874. https://www.usenix.org/conference/usenixsecurity22/presentation/dai

[10] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: A metric for measuring information leakage. In *2012 39th Annual International Symposium on Computer Architecture ISCA)*. 106–117. https://doi.org/10.1109/ISCA.2012.6237010

[11] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2019. Analysis of secure caches using a three-step model for timing-based attacks. *Journal of Hardware and Systems Security* 3, 4 (2019), 397–425.

[12] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2020. A Benchmark Suite for Evaluating Caches' Vulnerability to Timing Attacks. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS)*. ACM. https://doi.org/10.1145/3373376.3378510

[13] Peter W Deutsch, Yuheng Yang, Thomas Bourgeat, Jules Drean, Joel S Emer, and Mengjia Yan. 2022. DAGguise: mitigating memory timing side channels. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 329–343.

[14] Leonid Domnitser, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2010. A Predictive Model for Cache-Based Side Channels in Multicore and Multithreaded Microprocessors. In *Computer Network Security*, Igor Kotenko and Victor Skormin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 70–85.

[15] 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 Trans. Archit. Code Optim.* 8, 4, Article 35 (jan 2012), 21 pages. https://doi.org/10.1145/2086696.2086714

[16] Goran Doychev, Dominik Feld, Boris Kopf, Laurent Mauborgne, and Jan Reineke. 2013. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. In *22nd USENIX Security Symposium USENIX Security 13)*. USENIX Association, Washington, D.C., 431–446. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/doychev

[17] Andrew Ferraiuolo, Rui Xu, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. 2017. Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis. *SIGPLAN Not.* 52, 4 (apr 2017), 555–568. https://doi.org/10.1145/3093336.3037739

[18] Andrew Ferraiuolo, Mark Zhao, Andrew C. Myers, and G. Edward Suh. 2018. HyperFlow: A Processor Architecture for Nonmalleable, Timing-Safe Information Flow Security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1583–1600. https://doi.org/10.1145/3243734.3243743

[19] Hazem Gamal. 2020. Document Distance. https://github.com/Hazem-Gamall/document-distance.

[20] Daniel Genkin, William Kosasih, Fangfei Liu, Anna Trikalinou, Thomas Unterluggauer, and Yuval Yarom. 2022. CacheFX: A Framework for Evaluating Cache Security. https://doi.org/10.48550/ARXIV.2201.11377

[21] Zecheng He and Ruby B Lee. 2017. How secure is your cache against side-channel attacks?. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 341–353.

[22] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding contention-based channels and using them for defense. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture HPCA)*. 639–650. https://doi.org/10.1109/HPCA.2015.7056069

[23] Ibrahim Issa, Sudeep Kamath, and Aaron B. Wagner. 2016. An operational measure of information leakage. In *2016 Annual Conference on Information Science and Systems CISS)*. 234–239. https://doi.org/10.1109/CISS.2016.7460507

[24] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In *51st Annual IEEE/ACM International Symposium on Microarchitecture MICRO)*. IEEE. https://doi.org/10.1109/MICRO.2018.00083

[25] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy SP)*. 1–19. https://doi.org/10.1109/SP.2019.00002

[26] 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* (Alexandria, Virginia, USA) *CCS '07)*. Association for Computing Machinery, New York, NY, USA, 286–296. https://doi.org/10.1145/1315245.1315282

[27] Boris Köpf and David Basin. 2011. Automatically Deriving Information-Theoretic Bounds for Adaptive Side-Channel Attacks. *J. Comput. Secur.* 19, 1 (jan 2011), 1–31.

[28] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).

[29] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy SP)*. IEEE. https://doi.org/10.1109/SP.2015.43

[30] Mulong Luo, Andrew C. Myers, and G. Edward Suh. 2020. Stealthy Tracking of Autonomous Vehicles with Cache Side Channels. In *29th USENIX Security Symposium USENIX Security 20)*. USENIX Association, 859–876. https://www.usenix.org/conference/usenixsecurity20/presentation/luo

[31] Robert Martin, John Demme, and Simha Sethumadhavan. 2012. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *2012 39th Annual International Symposium on Computer Architecture ISCA)*. IEEE, 118–129.

[32] Michael Neve and Jean-Pierre Seifert. 2006. Advances on Access-driven Cache Attacks on AES. In *Selected Areas in Cryptography*. Springer.

[33] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *25th USENIX Security Symposium USENIX Security)*. USENIX Association.

[34] Antoon Purnal and Ingrid Verbauwhede. 2019. Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache. *CoRR* abs/1908.03383 (2019). arXiv:1908.03383 http://arxiv.org/abs/1908.03383

[35] Moinuddin K Qureshi. 2019. New attacks and defense for encrypted-address cache. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture ISCA)*. IEEE, 360–371.

[36] Gururaj Saileshwar and Moinuddin Qureshi. 2021. MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design. In *30th USENIX Security Symposium USENIX Security 21)*. USENIX Association, 1379–1396. https://www.usenix.org/conference/usenixsecurity21/presentation/saileshwar

[37] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news* 41, 3 (2013), 475–486.

[38] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. 2015. Avoiding Information Leakage in the Memory Controller with Fixed Service Policies. In *Proceedings of the 48th International Symposium on Microarchitecture MICRO)*. ACM. https://doi.org/10.1145/2830772.2830795

[39] Aria Shahverdi, Mahammad Shirinov, and Dana Dachman-Soled. 2021. Database Reconstruction from Noisy Volumes: A Cache Side-Channel Attack on SQLite. In *30th USENIX Security Symposium USENIX Security 21)*. USENIX Association, 1019–1035. https://www.usenix.org/conference/usenixsecurity21/presentation/shahverdi

[40] Anatoly Shusterman, Zohar Avraham, Eliezer Croitoru, Yarden Haskal, Lachlan Kang, Dvir Levi, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2021. Website Fingerprinting Through the Cache Occupancy Channel and its Real World Practicality. *IEEE Transactions on Dependable and Secure Computing* 18, 5 (2021), 2042–2060. https://doi.org/10.1109/TDSC.2020.2988369

[41] Jakub Szefer. 2016. Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses. *Journal of Hardware and Systems Security* (2016).

[42] Daniel Townley, Kerem Arıkan, Yu David Liu, Dmitry Ponomarev, and Oguz Ergin. 2022. Composable Cachelets: Protecting Enclaves from Cache Side-Channel Attacks. In *2022 USENIX Security Symposium*.

[43] 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* (Chicago, Illinois, USA) *CCSW '11)*. Association for Computing Machinery, New York, NY, USA, 41–46. https://doi.org/10.1145/2046660.2046671

[44] Paulo Barreto Vincent Rijmen, Antoon Bosselaers. [n. d.]. *Optimised ANSI C code for the Rijndael cipher*. https://opensource.apple.com/source/BerkeleyDB/BerkeleyDB-15/db/crypto/rijndael/rijndael-alg-fst.c.auto.html

[45] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. 2014. Timing Channel Protection for a Shared Memory Controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture HPCA)*. IEEE. https://doi.org/10.1109/HPCA.2014.6835934

[46] Zhenghong Wang and Ruby B Lee. 2007. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th annual international symposium on Computer architecture*. 494–505.

[47] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. 2013. SurfNoC: A Low Latency and Provably Non-interfering Approach to Secure Networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture ISCA)*.

ACM. https://doi.org/10.1145/2485922.2485972

[48] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. 2019. {ScatterCache}: Thwarting Cache Attacks via Cache Set Randomization. In *28th USENIX Security Symposium USENIX Security 19)*. 675–692.

[49] Benjamin Wu, Aaron B. Wagner, and G. Edward Suh. 2020. A Case for Maximal Leakage as a Side Channel Leakage Metric. *CoRR* abs/2004.08035 (2020). arXiv:2004.08035 https://arxiv.org/abs/2004.08035

[50] 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)*. USENIX Association.

[51] Yuanyuan Yuan, Qi Pang, and Shuai Wang. 2022. Automated Side Channel Analysis of Media Software with Manifold Learning. In *31st USENIX Security Symposium USENIX Security 22)*. USENIX Association, Boston, MA, 4419–4436. https://www.usenix.org/conference/usenixsecurity22/presentation/yuan-yuanyuan

[52] 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* (New Orleans, Louisiana, USA) *ACSAC '14)*. Association for Computing Machinery, New York, NY, USA, 96–105. https://doi.org/10.1145/2664243.2664273

[53] 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* (Tel-Aviv, Israel) *HASP '13)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/2487726.2487728

[54] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. 2018. Analyzing cache side channels using deep neural networks. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 174–186.

[55] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. 2017. Camouflage: Memory Traffic Shaping to Mitigate Timing Attacks. In *2017 IEEE International Symposium on High Performance Computer Architecture HPCA)*. IEEE. https://doi.org/10.1109/HPCA.2017.36