Security Analysis of Cache Replacement Policies

Pablo Cañones[1], Boris Köpf[1], and Jan Reineke[2]

[1] IMDEA Software Institute, Madrid, Spain
{pablo.canones,boris.koepf}@imdea.org

[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
reineke@cs.uni-saarland.de

Abstract. Modern computer architectures share physical resources between different programs in order to increase area-, energy-, and cost-efficiency. Unfortunately, sharing often gives rise to side channels that can be exploited for extracting or transmitting sensitive information. We currently lack techniques for systematic reasoning about this interplay between security and efficiency. In particular, there is no established way for quantifying security properties of shared caches.

In this paper, we propose a novel model that enables us to characterize important security properties of caches. Our model encompasses two aspects: (1) The amount of information that can be *absorbed* by a cache, and (2) the amount of information that can effectively be *extracted* from the cache by an adversary. We use our model to compute both quantities for common cache replacement policies (FIFO, LRU, and PLRU) and to compare their isolation properties. We further show how our model for information extraction leads to an algorithm that can be used to improve the bounds delivered by the CacheAudit static analyzer.

1 Introduction

Modern computer architectures share physical resources across different programs in order to increase area-, energy-, and cost-efficiency. Examples of commonly shared resources are caches, branch prediction units, DRAM, and disks.

Unfortunately, sharing poses a threat to security: even if programs are completely isolated on a logical level, sharing a physical resource usually means that one program's resource usage pattern can be observed by the other. This constitutes a channel that can be exploited for extracting or transmitting sensitive information. While this kind of vulnerability has been known for decades [14], its severity has become painfully apparent with a stream of highly effective side-channel attacks. One shared resource that has been the objective of a large number of attacks are CPU caches, e.g. [20,6,2,3,12,24,16].

From a security point of view it would be ideal to completely eliminate side channels through the cache by design, as in [22,25], or to flush the cache between accesses of two different parties. Unfortunately, such conservative approaches also partially void the performance benefits of sharing. In many practical scenarios, designers will opt for less conservative solutions that offer "sufficient" degrees of security together with high performance. However, while there is a large body of work on evaluating the impact of different cache designs on performance, there are no established metrics for evaluating their security, which prevents principled decision-making in that design space.

Approach. In this paper, we address this problem by introducing a novel approach to quantify the security of caches, in particular: their replacement policies. Our approach aims to answer the following questions, which capture two natural aspects of isolation between programs that share the cache:

Q1 *How much information about a computation is absorbed by the cache?* There are two challenges involved with this question. The first is to identify a meaningful measure for the information contained in a given cache state. The second is to characterize the set of possible computations, which may induce different cache states. To make assertions about the security of the cache architecture (rather than about the security of a specific program running on top of a cache architecture) such a characterization needs to encompass a sufficiently general class of programs.

Q2 *How much information can an adversary extract from the cache state?* The challenge for answering this question is that an adversary can only learn about the cache state by probing, that is, by performing memory accesses and measuring their latency. However, probing also modifies the cache state and thus can reduce its information content. With the exception of one approach that encompasses secrets that change over time [17], existing models of quantitative information flow do not account for this scenario because they either consider only single probes [21] or assume the secret remains unchanged by the probing [13,7,4].

A1 For answering Q1, we characterize the absorbed information as the number of reachable cache states, which essentially captures the information that programs leak *into* the cache. For a single program, this amount can be bounded using existing static analysis tools [10]. For abstracting from a specific program, we draw inspiration from the working set model [9] and characterize programs in terms of their footprint, i.e., the number of memory blocks they use. We then show how (and under which assumptions) the footprint alone can be used to characterize the absorption of a given replacement policy, leading to a program-independent measure.

A2 For answering Q2, we put forward a novel model to quantify the "extractable" information about the cache state. We consider an adversary that adaptively provides inputs and observes the outputs. The key difference to existing models of adaptive attacks [13,7] is that our model is based on a Mealy machine in which each input triggers a state transition, which may erase information about its origin. As in existing models, we first characterize the revealed information in terms of a partition of the set of secrets (here: initial states of the machine). We then evaluate this partition with established measures of leakage to quantify the corresponding amount of information. By considering the maximum leakage w.r.t. all possible inputs to the Mealy machine, we obtain an upper bound on the information that any adaptive adversary can extract. We present an algorithm that computes such bounds for given Mealy machines.

Results. We put our models and algorithms to work for the quantification of absorption and extraction properties of common cache replacement policies, namely FIFO, LRU, and PLRU. We highlight the following results; see the paper for more details.

– We show that the relative security ranking of cache replacement policies varies widely depending on the memory demand of the program. For example, FIFO can provide the best security when memory demand is low, whereas LRU generally provides the best security. Our results show that PLRU generally offers worse security than the other replacement policies.
– We show that our algorithm for information extraction can be used for improving the cache-state counting of the CacheAudit static analyzer [10]. Our experimental results show that this significantly improves the bounds delivered by CacheAudit, leading to gains of up to 50 bits for AES 256.

Contribution. In summary, our conceptual contribution is to propose novel measures for quantifying isolation properties of shared caches. Our practical contribution is to perform the first security analysis of common cache replacement policies.

2 The Model

2.1 Caches as Mealy Machines

Caches are fast but small memories that store a subset of the main memory's contents to bridge the latency gap between the CPU and the main memory. To profit from spatial locality and to reduce management overhead, main memory is logically partitioned into a set $\mathcal{B}$ of memory blocks. Each block is cached as a whole in a cache line of the same size. When accessing a memory block, the cache logic has to determine whether the block is stored in the cache ("cache hit") or not ("cache miss").

In this paper, we model caches as Mealy machines, that is, finite automata that map sequences of accessed memory blocks to sequences of hits and misses. We begin by recalling the definition of a Mealy machine before we specialize it to the case of caches.

Definition 1. *A (deterministic)* Mealy machine *M is a five-tuple consisting of*

– *S: a finite set of* states*,*
– *Σ: a finite set of* inputs*,*
– *O: a finite set of* outputs *(or* observations*),*
– *$upd\colon S \times \Sigma \to S$: a* transition function*, and*
– *$view\colon S \times \Sigma \to O$: an* observation function

For casting caches as Mealy machines, we use memory blocks as inputs, i.e. $\Sigma = \mathcal{B}$, and cache hits (H) and misses (M) as observations, i.e., $O = \{\mathsf{H}, \mathsf{M}\}$. For defining the set of states S, recall that caches are commonly partitioned into independent equally-sized *cache sets* whose size A is called the *associativity* of the cache. For each block there is a single cache set that stores it.

For simplicity of presentation we focus on caches with a single set. Since cache sets behave independently from each other, the technique is generalizable to several sets by focusing each time on the blocks stored in a particular set. We model a cache set as a function that assigns an age in $\mathcal{A} := \{0, \ldots, A-1, A\}$ to each memory block.

$$S = \{c \in \mathcal{B} \to \mathcal{A} \mid \ \forall b_1, b_2 \in \mathcal{B} : b_1 \neq b_2 \ \Rightarrow c(b_1) \neq c(b_2) \lor c(b_1) = c(b_2) = A)\} \ .$$

Here, the youngest block has age 0 and the oldest cached block has age $A-1$. Age A means that a block is not cached; it is the only age that can be shared by multiple blocks.

With this, the observation function $\mathit{view}_b = \mathit{view}(\cdot, b)$ is naturally defined as

$$\mathit{view}_b(c) = \begin{cases} \mathsf{H} & \text{if } c(b) < A \\ \mathsf{M} & \text{else} \end{cases}$$

The transition function $\mathit{upd}_b = \mathit{upd}(\cdot, b)$ is specified by:

$$\mathit{upd}_b(c)(b') = \begin{cases} c(b') & \text{if } b' \neq b \land c(b') = A \\ 0 & \text{if } b' = b \land c(b) = A \\ c(b') + 1 & \text{if } b' \neq b \land c(b') < A \land c(b) = A \\ \Pi_{c(b)}(c(b')) & \text{if } c(b') < A \land c(b) < A \end{cases} \qquad (1)$$

This transition function models *permutation* replacement policies as defined in [1]. Upon a miss, $c(b) = A$, the accessed block is placed at the beginning of the cache, increasing the ages of younger blocks and evicting the block with age $A-1$. In the case of a hit, each replacement policy reorders the blocks in a certain way, determined by the *permutation* function $\Pi_\alpha(\alpha') : \mathcal{A} \to \mathcal{A}$; it modifies the current age α' of a block according to a *base* age α.

Each replacement policy has its own permutation function: FIFO does not reorder the blocks, LRU sets the age of the accessed block to 0, and PLRU behaves similar to LRU but with a more complex reorganization. We refer to the Mealy machines corresponding to LRU, PLRU, and FIFO caches by $M_{\mathrm{LRU}}, M_{\mathrm{PLRU}}$, and M_{FIFO}, respectively.

The formalization of these policies, as well as the proofs of all technical results are contained in the extended version of this paper [8].

2.2 Quantifying Absorption and Extraction

We characterize absorption and extraction in terms of the interactions of two agents, a victim and an adversary.

– The *victim* first chooses a secret, such as a cryptographic key. We model this using a random variable X. The victim then uses this secret as input to a program that he runs to completion (or preemption) on a platform with a cache. We capture the effect of the victim's computation on the cache state in terms of a finite sequence of blocks from the set of *victim's blocks* B_v, where $B_v \subseteq \mathcal{B}$. The cache uses this sequence as inputs to transition from an initial state to the *victim's state*. We model the victim's state using a random variable Y_v that takes values in a set $S_v \subseteq S$, i.e. $ran(Y_v) = S_v$.

– The *adversary* then runs a program on the same platform, which enables him to make observations about the state of the cache by measuring the latency of its memory accesses.[3] We model the adversary's actions in terms of a finite sequence of blocks from the subset of *attacker's blocks* $B_a \subseteq \mathcal{B}$. Using the sequence of blocks as inputs, the cache transitions from the victim's state returning a sequence of hits and misses that we model with the random variable Z_a, $ran(Z_a) \subseteq O^*$. We make the random variable dependent on the attacker since he can choose the sequence of blocks. Based on these observations, the adversary tries to guess the secret. We model the guess in terms of the random variable $\hat{X}$.[4] We say that an attack is successful if the adversary correctly guesses the secret, i.e. if $X = \hat{X}$.

We now give a high-level operational motivation for our definitions of information absorption and extraction, in terms of a bound on the probability of a successful attack. We assume that the distribution of each of these random variables depends only on the outcome of the previous one, i.e., that the distribution of cache states depends only on the secret, and that the adversary's observations depend only on the state of the cache. Then we can cast the dependencies between these random variables in terms of the following Markov chain:

$$\begin{tikzcd} \underset{\text{Secret}}{X} \arrow[r, "\overset{\text{Victim}}{|}"] & \underset{\text{Cache State}}{Y_v} \arrow[r, "\overset{\text{Adversary probe}}{|}"] & \underset{\text{Observation}}{Z_a} \arrow[r, "\overset{\text{Adversary guess}}{|}"] & \underset{\text{Guess}}{\hat{X}} \end{tikzcd} \tag{2}$$

The following result bounds the probability of a successful attack, i.e. $P(X = \hat{X})$, in terms of the size of the ranges of Y_v and Z_a, respectively.

Theorem 1.

$$P(X = \hat{X}) \leq \max_{x \in ran(X)} P(X = x) \cdot |ran(Z_a)| \tag{3}$$

$$P(X = \hat{X}) \leq \max_{x \in ran(X)} P(X = x) \cdot |ran(Y_v)| \tag{4}$$

For an attacker that follows a deterministic strategy, the value of Z_a is determined by the value of Y_v. Therefore $|ran(Z_a)| \leq |ran(Y_v)|$, which implies that (3) leads to better security guarantees than (4).

[3] In the literature, this is known as an *access-based adversary*, e.g. [19].

[4] Note that, while Y_v and Z_a are given in terms of inputs and outputs of the Mealy machine representing the cache, we do not assume any particular structure on X and $\hat{X}$.

Whenever additionally the value of Y_v is determined by that of X and X is uniformly distributed, the bounds given by Theorem 1 are tight, in the sense that they can be achieved by computationally unbounded adversaries.

In this paper, we will use $|ran(Y_v)|$ to capture the amount of information that is *absorbed* by the cache, and we will use $|ran(Z_a)|$ to capture the amount of information that the adversary can *extract* from the cache. The operational significance of these quantities follows from Theorem 1. We discuss how these quantities can be computed in Section 3 and 4, respectively.

3 Absorption of Information

In this section we characterize the information absorption of different cache replacement policies. That is, we characterize $ran(Y_v)$ from (2) as a subset $S_v \subseteq S$ of reachable victim's states of the Mealy machine representing the cache.

Before we give the formal definition we note that the absorbed information depends on two things: the initial state of the Mealy machine and the inputs of the victim. To see the effect of the initial state $s_0 \in S$, consider the Mealy machine in Figure 1 and assume that the victim may use any sequence of inputs from $\Sigma_v^* = \{a, b\}^*$. If we start from the state $s_0 = 1$ only that one state is reachable, $S_v = \{1\}$; if $s_0 = 2, 3$ then $S_v = \{1, 2, 3\}$ and finally if $s_0 = 4$ then $S_v = S$.

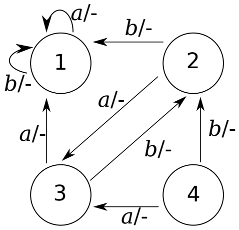


Fig. 1: Example of Mealy machine

We capture the victim's inputs as a trace $t \in \Sigma_v^*$. This leads to the following definition of $|ran(Y_v)|$.

Definition 2. *We define the* absorbed information *of a Mealy machine* $M = (S, \Sigma, O, upd, view)$ *w.r.t an initial state* s_0 *and a set of traces* $T \subseteq \Sigma_v^*$ *as*

$$Abs(M, s_0, T) = |\{s \in S \mid \exists t \in T \colon upd_t(s_0) = s\}| ,$$

In the above definition of absorption, the set of traces T is a parameter. For a given program, existing static analysis techniques can be used to compute approximations of the set of traces T and the induced absorption of a particular cache, modeled by a Mealy machine M. In Section 6 we present the results of a static analysis of two AES implementations.

In this section, our goal is to characterize the absorption properties of caches *independently* of a particular program. A worst case approach to this end is to study absorption under all possible traces $T = B_v^*$, given a set of memory blocks B_v. For this, we first state several general results in Section 3.1, which show that the absorption of caches is independent of the particular set of memory blocks B_v being accessed, and only depends on its size, $|B_v|$. In Section 3.2, we then use these general results to derive concrete results on the absorption properties of caches under LRU, FIFO, and PLRU replacement.

3.1 Data Independence of Permutation Replacement Policies

Initial State Absorption, as defined in Definition 2 depends on the initial state of the Mealy machine. Considering programs that may access the set of memory blocks $B \subseteq \mathcal{B}$, two types of initial states for caches are particularly interesting:

Definition 3. *We say that a cache state $c\colon \mathcal{B} \to \mathcal{A}$ is*

1. empty w.r.t. B *if* $c(B) = \{c(b) \mid b \in B\} = \{A\}$. *That is, none of the blocks in B are cached.*
2. filled with B *if* $c(B) = \{0, \ldots, \min(A, |B| - 1)\}$. *That is, the blocks in B occupy the cache. If B contains less blocks than cache lines, we require that the first $|B|$ lines are filled.*

The notions of *empty* and *filled* cache states are relative to a set of memory blocks. We will consider empty and filled cache states relative to the memory blocks accessed by the victim, B_v. To conservatively capture the power of an attacker, ages without a victim's block mapped to them will be assumed to hold the attacker's memory blocks not accessible for the victim, that is, blocks from the set $B_a \setminus B_v$.

Data Independence The following result is central for our program-independent analysis of cache replacement policies. It shows that absorption can be characterized independently of the particular set of blocks B that the victim may access:

Theorem 2. *Whenever $|B_1| = |B_2|$, and c_1 is empty (filled) w.r.t. B_1 and c_2 empty (filled) w.r.t. B_2, then*

$$Abs(M, c_1, B_1^*) = Abs(M, c_2, B_2^*).$$

The proof of Theorem 2 follows from the following lemma and the observation that one can define bijections between all sets of equal cardinality.

Lemma 1. *Let $f\colon \mathcal{B} \to \mathcal{B}$ be a bijection. Then*

$$Abs(M, c_0, B^*) = Abs(M, c_0 \circ f^{-1}, (f(B))^*).$$

We focus on filled and empty initial states since they represent the two extremes for the information absorption. Consider a *partially filled* state c, that is, where there is a sequence of distinct blocks $b_0 \ldots b_n$ with $n \leq \min(A, |B| - 1)$ such that $c(b_i) = i$ for $i \leq n$. Then, any state reachable from c by inputting a trace $t \in B^*$ is reachable from an empty one c_e with the trace $t' = b_n \ldots b_0 t$. Since c_e is empty, we load the blocks $b_0 \ldots b_n$ in reverse order; these access produce misses and so, after the updates, $upd_{b_0} \cdots upd_{b_n}(c_e)(b_i) = i$, see (1). Therefore $Abs(M, c, B^*) \leq Abs(M, c_e, B^*)$. Using this argument we can see that, for the same set of memory blocks, the value of the absorbed information is the smallest when starting on a filled state and is the largest when starting on an empty state.

An important consequence of Theorem 2 is that, given an identical status, i.e. empty or filled, of the initial state, the amount of absorbed information depends only on the number of blocks in B_v. We call this number the *footprint* and denote it by $\mathit{fp} = |B_v|$. This terminology is loosely connected with the notion of a memory footprint as used in the theory of locality [23]. Theory of locality defines the footprint as the number of distinct memory blocks accessed during a time window, i.e. on a trace of a given length. In our case we consider this length to be unbounded so the trace is the whole execution of the program. This motivates the specialization of the definition of the absorbed information in terms of the footprint, namely

$$\mathit{Abs}_x(M, \mathit{fp}) = \mathit{Abs}(M, c_0, (B_v)^*) \ ,$$

where we use the subscript $x = e$ to denote that c_0 is empty w.r.t. B_v, and $x = f$ to denote that c_0 is filled w.r.t. B_v.

3.2 Analysis of Cache Replacement Policies

Next we give a summary of our program-independent analysis of the absorption for each replacement policy.

Results for Filled Caches For some replacement policies, when the cache is filled and the footprint is small enough, some cache states are unreachable from the initial state, which reduces the information absorption. The details for each policy are given below. In case *every* state of the cache is reachable, we count all the possible feasible mappings of *fp* blocks to the set of ages $\mathcal{A}$. Then the absorbed information is the number of *k-permutations of n* of the memory blocks, i.e., the number of different ordered arrangements of *fp* blocks in a sequence of up to A elements.

Proposition 1. *For M_{LRU}, the absorbed information for a filled cache is:*

$$\mathit{Abs}_f(M_{LRU}, \mathit{fp}) = \begin{cases} \mathit{fp}! & \text{if } \mathit{fp} < A, \\ \frac{\mathit{fp}!}{(\mathit{fp}-A)!} & \text{if } \mathit{fp} \geq A. \end{cases}$$

Proposition 2. *For M_{FIFO}, the absorbed information for a filled cache is:*

$$\mathit{Abs}_f(M_{FIFO}, \mathit{fp}) = \begin{cases} 1 & \text{if } \mathit{fp} \leq A, \\ A+1 & \text{if } \mathit{fp} = A+1, \\ \frac{\mathit{fp}!}{(\mathit{fp}-A)!} & \text{if } \mathit{fp} > A+1. \end{cases}$$

Proposition 3. *For M_{PLRU}, the absorbed information for a filled cache is:*

$$\mathit{Abs}_f(M_{PLRU}, \mathit{fp}) = \begin{cases} 2^{\mathit{fp}-1} & \text{if } 1 \leq \mathit{fp} \leq A, \\ \frac{\mathit{fp}!}{(\mathit{fp}-A)!} & \text{if } \mathit{fp} > A. \end{cases}$$

Results for Empty Caches The case of an empty cache is more complex to analyze. First we need to explain a special behavior of PLRU that produces extra reachable states which increases its absorption with respect to the other two policies.

Example 1. Consider a 4-way cache that starts in a state consisting of the attacker's blocks $\{x_0, x_1, x_2, x_3\} \subseteq B_a$ where we are going to access three victim blocks in a specific order, $a, b, c \in B_v$. For any of the three replacement policies the state becomes:

$$[x_0, x_1, x_2, x_3] \underset{a}{\rightsquigarrow} [a, x_0, x_1, x_2] \underset{b}{\rightsquigarrow} [b, a, x_0, x_1] \underset{c}{\rightsquigarrow} [c, b, a, x_0],$$

where the leftmost element of the lists has age zero and the one on the right is the oldest. Consider that we now access block b again. The cache states transition to: $[b, c, a, x_0]$ for LRU, $[c, b, a, x_0]$ for FIFO and $[b, c, x_0, a]$ for PLRU (note the age of the last attacker's block x_0). The state obtained by PLRU is unreachable for the other two replacement policies, since they always fill up the cache consecutively from left to right. This illustrates how the information absorption for PLRU is larger than for the other policies.

The example is independent of the blocks being used but a consequence of the fact that we are inputting $k < A$ blocks. For LRU and FIFO, any sequence using $k < A$ victim blocks will transform an initial state $[x_0, x_1, \ldots, x_{A-1}]$ to a state of the form $[\_, \ldots, \_, x_0, \ldots, x_{A-1-k}]$, where victim blocks are denoted by "_". In the case of PLRU this is not always the case, as the previous example shows.

Following our definition of absorption, we assume that the victim may input any sequence of blocks. Then the number of reachable cache states can be determined as follows:

1. Determine the set of reachable *configurations*, i.e., cache states in which the victim's memory blocks are not distinguished from each other, but instead represented by the *placeholder* "_".
2. Determine for each configuration the number of concrete cache states the configuration represents, i.e., the number of ways the victim's blocks may fill its placeholders.

This procedure can further be simplified upon by the following observation: The number of concrete cache states that a configuration represents, only depends on its number of placeholders and the number of victim blocks to consider: Given k placeholders and $fp \geq k$ victim's memory blocks, a configuration represents exactly $\frac{fp!}{(fp-k)!}$ cache states.

Let $\Lambda_M(k, A)$ denote the number of reachable configurations under policy M, associativity A, with exactly k placeholders. Accessing fp distinct memory blocks may yield configurations with 0 to fp many placeholders. Based on this notion, we obtain the following general characterization of a replacement policy's absorption:

Proposition 4. *For any replacement policy M, the absorbed information starting from an empty cache is:*

$$Abs_e(M, fp) = \sum_{k=0}^{\min\{fp, A\}} \Lambda_M(k, A) \frac{fp!}{(fp-k)!}.$$

Lemma 2. *For LRU and FIFO, $\Lambda_M(k, A) = 1$ for any number of placeholders k and associativity A. For PLRU, $\Lambda_{M_{PLRU}}(k, A)$ is given by:*

$$\Lambda_{M_{PLRU}}(k, A) = 2 \cdot \sum_{i=\max\{1, k-\frac{A}{2}\}}^{\min\{\frac{A}{2}, k-1\}} \Lambda_{M_{PLRU}}(i, \tfrac{A}{2}) \cdot \Lambda_{M_{PLRU}}(k-i, \tfrac{A}{2}), \tag{5}$$

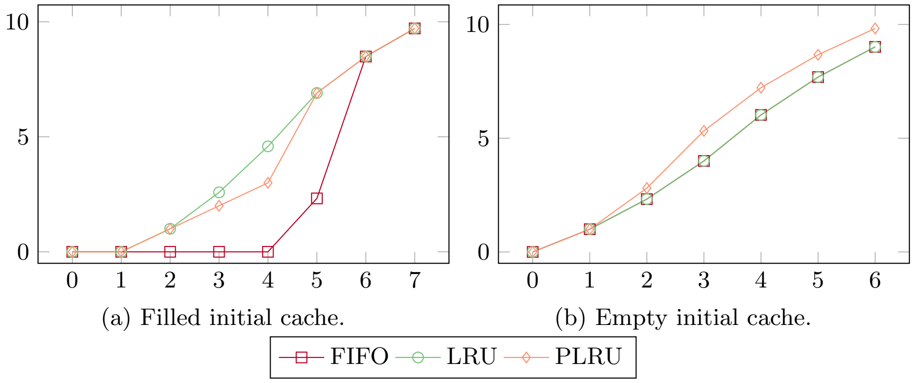
if $1 < k < A$ and $\Lambda_{M_{PLRU}}(k, A) = 1$ if $k \leq 1$ or $k = A$.

Comparison of Absorption Let us compare the absorption of LRU, FIFO, and PLRU based on Propositions 1 to 4, for a cache set of associativity 4. Similar results can be obtained for any associativity. The results depicted in Figure 2 can be obtained both from the formulas above or by simulation of caches. We highlight the following observations.

- For each replacement policy, the absorbed information grows monotonically with the footprint, as expected.
- The absorption for an empty initial state is always larger than for a filled state.
- For a filled initial state, LRU absorbs always at least as much information as the other replacement policies since every state is always reachable. For large enough footprints, the absorption coincides for all policies.
- For an empty initial state PLRU absorbs most. This is due to the fact that PLRU may leave "holes" in the cache state, see Example 1.
- For a filled initial cache, FIFO does not absorb any information, whenever the footprint is smaller than the associativity. This captures the intuition that preloading of sensitive data can increase security, as long as all data fits into the cache. In case it does not, the positive effect of preloading is, however, quickly undone.

4 Extraction of Information

In this section we characterize the information extraction for different cache replacement policies. That is, we characterize $ran(Z_a)$ from (2). For this we develop a novel model that characterizes the information an adaptive attacker can learn about the initial state of a Mealy machine. We then use the model to derive bounds on the information that can be extracted from caches with different replacement policies.



(a) Filled initial cache.

(b) Empty initial cache.

Fig. 2: Information absorption of a 4-way cache set. Figure 2a depicts the case of a filled initial cache, part 2b an empty one. In both figures, the horizontal axis depicts the footprint, i.e., the number of memory blocks used. The vertical axis depicts the absorbed information on a logarithmic scale, that is, in *bits*. Note that in Figure 2b, the line for LRU and FIFO coincides.

4.1 Probing Strategies

Let $M = (S, \Sigma, O, upd, view)$ be a Mealy machine. A *probe* p of M is an alternating sequence $p = \sigma_1 o_1 \sigma_2 \dots \sigma_n o_n$ of inputs $\sigma_i \in \Sigma_a \subseteq \Sigma$ and observations $o_i \in O$, such that M outputs $o_1 \dots o_i$ when the sequence $\sigma_1 \dots \sigma_i$ is the input. We say that a state $s \in S$ is *coherent* with probe p if, for all $i \in \{1, \dots, n\}$, we have

$$view_{\sigma_i} upd_{\sigma_{i-1}} \cdots upd_{\sigma_1}(s) = o_i \ ,$$

i.e., the probe does not exclude s as a potential initial state of M. Along the lines of [5,13], we define the adversary's *knowledge set* $K(p)$ about the initial state of M as the subset of possible states that are coherent with probe p.

$$K(p) = \{s \in S_v \mid s \text{ is coherent with } p\}$$

For convenience, we also define the adversary's *final knowledge set* $FK(p)$ as the set of states that M may be in after receiving the inputs and producing the outputs in the probe p:

$$FK(p) = \{upd_{\sigma_n} \cdots upd_{\sigma_1}(s) \mid s \in K(p)\}$$

An adversary may be able to choose inputs based on previous observations, that is, the probing can be adaptive. To model adaptivity we introduce probing strategies. A *probing strategy* is a function from a sequence of observations to an input symbol, $\text{att} : O^* \mapsto \Sigma_a$. This way, the first input to make comes from applying the function to the empty sequence, $\sigma_1 = \text{att}(\varepsilon)$, the second input is a function of the previous observation, $\sigma_2 = \text{att}(o_1)$, and so, for any i $\sigma_i = \text{att}(o_1 \dots o_{i-1})$. We say that p is a probe of att, if p may be obtained from the probing strategy att.

We now present a toy example that we will use through the section to illustrate the use of probing strategies.

Example 2. Consider a Mealy machine where $S = S_v = \Sigma_a = \{0, 1, \ldots, 6\}$, the observation and transition function are:

$$\mathit{view}_\sigma(s) = \begin{cases} 0 & \text{if } s < \sigma - 1, \\ 2 & \text{if } s \in [\sigma - 1, \sigma + 1], \\ 1 & \text{if } \sigma + 1 < s. \end{cases} \quad \mathit{upd}_\sigma(s) = \begin{cases} s + 1 & \text{if } s < \sigma, \\ s & \text{if } s \in [\sigma, \sigma + 1], \\ s - 1 & \text{if } \sigma + 1 < s. \end{cases}$$

Consider the probing strategy given by the function $\mathrm{att}(o_1 \ldots o_n) = 0 + \sum_{i=1}^{n} o_i$, which starts by inputting 0 and determines the next input based on the previous outputs. We will later see that att is a good probing strategy in this example.

By definition, we can apply a probing strategy indefinitely on sequences of arbitrary length and thus probe the Mealy machine indefinitely. However, at some point additional inputs are of no use, as the following definition characterizes.

Definition 4. *We say that a probe $p = \sigma_1 o_1 \sigma_2 \ldots \sigma_n o_n$ of probing strategy att is* depleted *w.r.t. to att, if for all probes q of att that are extensions of p, i.e., $q = p\sigma_{n+1} o_{n+1} \sigma_{n+2} \ldots \sigma_m o_m$, the knowledge sets are equal, i.e., $K(p) = K(q)$. We say a depleted probe $p = \sigma_1 o_1 \sigma_2 \ldots \sigma_n o_n$ is of* minimal length *when, a probe q made of a sub-sequence of it, $q = \sigma_{k_1} o_{k_1} \sigma_{k_2} \ldots \sigma_{k_i} o_{k_i}$ for any $i < n$, is not depleted.*

We next show that the knowledge sets of depleted probes of a probing strategy form a partition of the states of M. That is, the knowledge sets of distinct sequences are pairwise disjoint and their union contains all states.

Proposition 5. *Given a probing strategy att, the set of all knowledge sets produced by depleted probes w.r.t. att*

$$R_{att} = \{K(p) \mid \mathit{probe}\ p = \mathit{att}(\varepsilon) o_1 \ldots \mathit{att}(o_1 \ldots o_{n-1}) o_n \wedge p \text{ is depleted w.r.t. att}\},$$

is a partition of the set of possible states S_v.

Before starting the probing, the attacker knows that the victim's state is an element of the set S_v. As he makes inputs and refines the knowledge sets, he reduces the number of coherent states and thus learns information about the victim's initial state. As depleted probes correspond to unrefinable knowledge sets, there is no point in further queries once a probe is depleted.

When constructing a strategy, the attacker needs to consider all the possible outputs that he might observe when eventually applying his strategy. Once all the knowledge sets obtained from an attack strategy cannot be further refined by additional queries, the probes are depleted and the attacker has along the way obtained the finest partition of the set S_v under that strategy and all possible extensions.

0/0	1/1 **0**	2/2	3/3	4/4	5/5	6/6
0/0 **2**	1/1	2/1	3/2 **1**	4/3	5/4	6/5
0/1	1/2	2/1 **3**	3/2	4/2	5/3 **2**	6/4
0/1	1/2	2/2	3/3	4/2 **4**	5/3	6/3
0/1	1/2	2/2	3/3	4/3	5/4	6/3

Table 1: Partition from Example 3.

Example 3. Following Example 2 we apply the probing strategy to the set of possible states and obtain the partition shown in Table 1. Each row shows the knowledge sets before and after the elements are updated (left and right, respectively). The first row shows the initial knowledge set, i.e., S_v. The bold face 0 indicates the first input symbol, which partitions the initial knowledge set into two knowledge sets, corresponding to the two possible outputs of the Mealy machine on the input 0. For each resulting knowledge set, except for the singleton ones where the probes are depleted, the figure then indicates the next input following the probing strategy and how it partitions its knowledge set. After at most four inputs we obtain a partition of all singleton knowledge sets.

For every attack strategy there is a finite set of depleted probes of minimal length. We define $Z_a = Z_{\text{att}}$ from (2) as the random variable that captures the sequence of observations obtained when following probing strategy att until obtaining a depleted probe of minimal length. So $ran(Z_{\text{att}}) \subseteq O^*$ is the set of sequences of observations obtained from the depleted probes of minimal length of att. Every depleted probe corresponds to a knowledge set; so we can relate every element of $ran(Z_{\text{att}})$ to a knowledge set. Therefore, computing $|\, ran(Z_{\text{att}})|$ is equivalent to counting the number of knowledge sets in the partition induced by the strategy att.

Definition 5. *We say that a strategy att is* optimal *if the partition R_{att} it induces on a set of possible states S_v, has the maximal number of knowledge sets among all strategies. We call this number $r_{\max}$ the* maximum information leakage.

The strategy presented in Example 2 is actually optimal since no partition can be better than the one that produces singleton knowledge sets. On the other hand, the strategy $\text{att}(o_1 \dots o_n) = 1 + \sum_{i=1}^{n} o_i$ is not optimal since the first input, 1, is not able to distinguish the initial states 0 and 1, which are both updated to 1 as a result of the input, $upd_1(0) = upd_1(1) = 1$, and so they can not be distinguished by this strategy.

4.2 Information Extraction in Caches

Here we derive bounds on the maximum information leakage for the three replacement policies. We prove bounds for LRU and FIFO based on the associativity of the cache and prove that for PLRU this bound depends also on the footprint.

We consider two types of attackers in terms of their set of memory blocks.

- *Shared memory attacker.* The attacker's set of blocks includes the victim's ones, $B_v \subset B_a$.
- *Disjoint memory attacker.* The sets of blocks of the attacker and the victim are disjoint $B_v \cap B_a = \emptyset$.

Proposition 6. *Consider M_{LRU} and M_{FIFO} with associativity A and a shared memory attacker. The maximum information leakage on any set of states is bounded by 2^A for M_{LRU} and by $(A+1)!$ for M_{FIFO}.*

Proposition 7. *Consider M_{PLRU} with associativity $A \geq 4$*[5] *and a shared memory attacker. Let $r_{\max}(fp)$ be the maximum information leakage obtained with a given footprint $fp \geq A$. It holds that $r_{\max}(fp+1) \geq r_{\max}(fp) + 1$.*

In the case of associativity four for M_{PLRU} the maximum information leakage is increased by eight with every new memory block, this can be seen in Figures 3e and 3f. This result also implies that the maximum information leakage for PLRU is unbounded.

Proposition 8. *Consider M_{FIFO} and M_{LRU} with associativity A, and a disjoint memory attacker. The maximum information leakage on any set of states is bounded by $A + 1$.*

Proposition 9. *Consider M_{PLRU} with associativity A, footprint fp, and a disjoint memory attacker. The maximum information leakage is bounded by $\sum_{k=0}^{fp} \Lambda_{PLRU}(k, A)$ where $\Lambda_{PLRU}(k, A)$ is defined as in* (5).

5 An Algorithm for Information Extraction

In this section we present an algorithm for computing the maximum information leakage $r_{\max}$ for a given Mealy machine. The algorithm complements Propositions 6 to 9 in that it can deliver $r_{\max}$ for a specific set of states $S_v \subseteq S$ and an arbitrary Mealy machine. We use it later to compute extraction w.r.t. a given memory footprint, and to replace the engine for counting cache states in the CacheAudit static analyzer, leading to tighter bounds on the leakage.

In principle, our algorithm enumerates all attack strategies att and computes their partitions R_{att} by grouping states in S_v according to the corresponding observations. Additionally, we use two techniques for improving efficiency and ensuring termination:

– First, instead of maintaining the knowledge sets $K(p)$, for every probe p, we maintain the final knowledge set $FK(p)$. Using the final knowledge set enables us to track the number of original knowledge sets, as required for computing leakage. At the same time it enables re-use of the computation leading to $FK(p)$ across different strategies.

[5] Note that for associativity 2, PLRU and LRU coincide.

– Second, we need to identify cycles when refining partitions in order to ensure termination. We say that a probe q is *redundant* w.r.t another probe p, if $FK(pq) = FK(p)$. That is, the probe q does not further refine the (final) knowledge set of p. The probe q represents a cycle, which we detect by keeping track of already visited final knowledge sets.
The pseudocode is given in Algorithm 1. We next argue its correctness.

Algorithm 1: Partition function.

```
Partition($S, view, upd, \Sigma^a, \mathcal{S}$) Data: set of possible states $S$ (initially $S = S_v$),
        observation function $view$, transition function $upd$, set of attacker's
        inputs $\Sigma^a$, flag sets $\mathcal{S}$ (initially $\mathcal{S} = \emptyset$).
Result: number of knowledge sets $r_{\max}$ in the partition.
begin
    // Look for redundant sequences
    if $S \in \mathcal{S}$ then
        return 1;
    end
    $r_{\max} = 1$;
    foreach $\sigma \in \Sigma^a$ do
        // If the leakage is equal to the size of the set, finish
        if $r_{\max} = |S|$ then
            return $r_{max}$;
        end
        // If the partition is not refined save the set
        if $|view_\sigma(S)| = 1$ then
            $\mathcal{S}' = \mathcal{S} \cup \{S\}$;
        // If the partition is refined erase the saved sets
        else
            $\mathcal{S}' = \emptyset$;
        end
        foreach $o_i \in view_\sigma(S)$ do
            $S_i = \{s \in S \mid view_\sigma(s) = o_i\}$;                        // partition
            $S'_i = upd_\sigma(S_i)$;                                              // update
            $r_i$ = Partition($S'_i, view, upd, \Sigma^a, \mathcal{S}'$);           // recursion
        end
        // Increase the number of produced knowledge sets
        $r_{\max} = \max(r_{\max}, \sum_i r_i)$;
    end
    return $r_{max}$;
end
```

Proposition 10. *Given a Mealy machine $M = (S, \Sigma, O, upd, view)$, Algorithm 1 terminates and finds the maximum information leakage $r_{\max}$ for a set of possible states S_v.*

6 Experimental Results

6.1 Extraction (Program-independent)

We use two alternative approaches for the program-independent evaluation of extraction properties cache replacement policies. The first is to rely on the upper bounds of Propositions 6 to 9. The second is to apply the algorithm presented in Section 5 to a set of states that represent the absorbed information for a given footprint. We determine that set for each cache replacement policy by a simple fixpoint computation. This algorithmic approach is more precise because it takes the absorbed information as a baseline, but it comes at the expense of higher computational cost.

We obtain the following results by using Algorithm 1, where we consider a single 4-way cache set. Figure 3 depicts our data. We highlight the following results:

– For shared-memory adversaries, FIFO and LRU reach the bound on the maximum information leakage given in Proposition 6, which is independent of the footprint, see Figures 3a to 3d. In contrast, with PLRU the number of knowledge sets increases with the footprint as predicted by Proposition 7, see Figures 3e to 3f.

– For disjoint-memory adversaries and a filled initial state we always obtain zero leakage. For PLRU and a footprint of 2 or 3 some cache lines remain unoccupied. As before, these unoccupied lines trigger additional observations, which explain the bump in Figure 3e.

– We observe that FIFO exhibits the smallest difference between absorption and extraction among all policies, i.e. once absorbed, it is comparably easy to extract information from the cache, see Figures 3a to 3b. This is because FIFO does not reorder blocks upon hits, which makes systematic search for the cache state easier.

6.2 Extraction (Program-dependent)

We now use Algorithm 1 for computing the information that can be extracted from the cache state w.r.t. a specific program. For this, we use as a basis the set S_v of states output by the CacheAudit static analyzer, when run on an implementation of AES 256. In this example we use a cache consisting of several independent cache sets of associativity 4, blocks of 64 bytes and overall sizes of 4, 8, and 16 KB. We consider two cases, one that starts from a filled cache and one that starts from an empty cache.

The full results are given in Figure 4; here we highlight the following results.

– We obtain the bounds on the absorbed information corresponds to using the CacheAudit static analyzer. The difference between the absorbed information and the extractable information corresponds to the precision gained by the development in this paper. This gain is generally higher when sets contain more blocks, and reaches up to 50 bits for LRU on a 4K cache with empty initial state and a shared memory attacker, see Figure 4d. That is, our extraction algorithm is a simple but powerful replacement for the model counting algorithms in CacheAudit.

– The figures show a change in slope at different points. This is due to the fact that the leakage about the full cache state is computed as the product of the leakages about the individual sets. When increasing the cache size for a fixed program, the footprint in each of the sets reduces. The combined effect of considering more sets with smaller footprint each accounts for the change in slope.

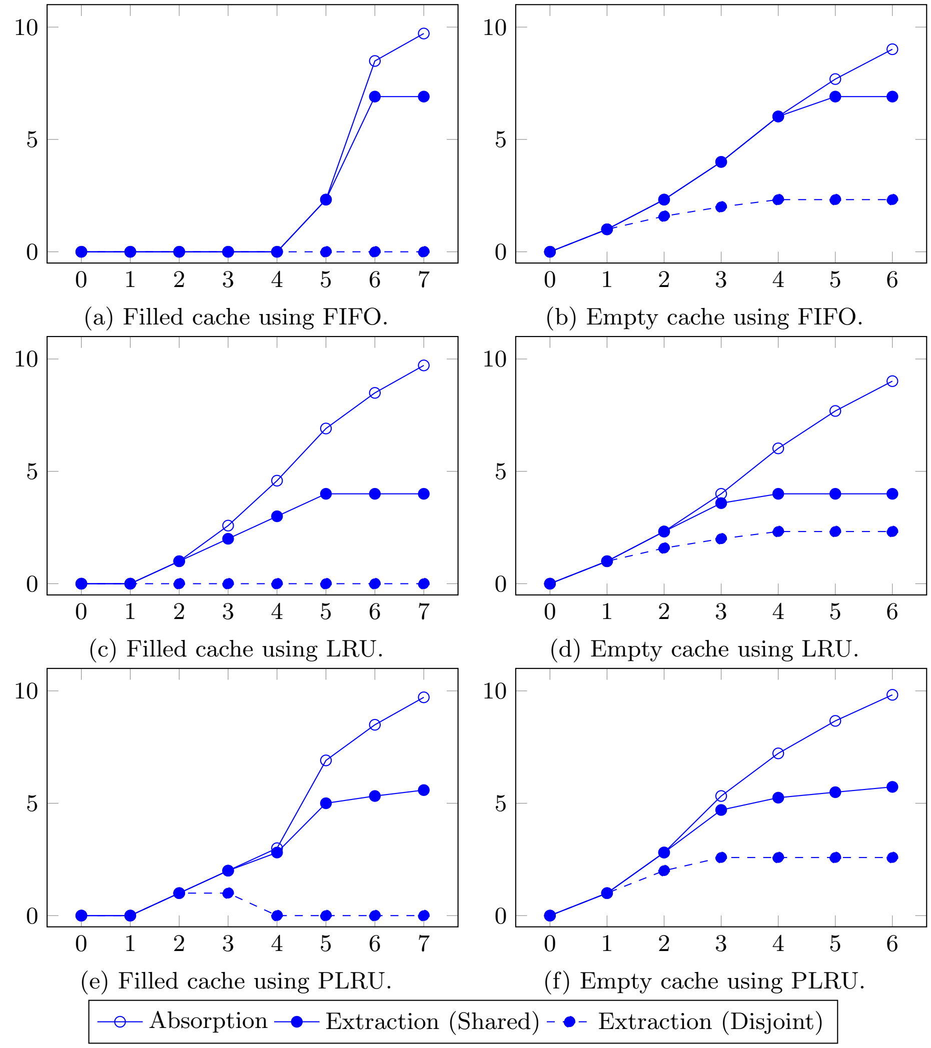
7 Related Work

Our work is related to existing models for adaptive probing [13,7]. There, however, the secret remains static. The model of [13] and the deterministic part of [7] is a special case of ours, where the update function is the identity.

Mardziel et al. [17] develop an approach to quantify information flow for *dynamic secrets*, that is, secrets that evolve over time. They consider a probabilistic system and attacks that consist of a fixed amount of steps. Attacks finish with an exploit whose success is evaluated using gain functions [4]. Our model for information extraction differs from their model in that it is deterministic and allows to compute leakage for an undetermined number of attacks steps, i.e., until the probing is depleted. We further provide an algorithm that actually allows us to compute optimal strategies. We leave a probabilistic extension of our model to future work.

The problem that we consider in this paper is related to the *state identification* problem for Mealy machines, which was first introduced by Moore in [18], expanded upon by Gill in [11], and analyzed from a complexity perspective by Lee and Yannakakis [15]. The state-identification problem is to determine the initial state of a Mealy machine by probing strategies, just as in our case. While we are interested in the maximal number of knowledge sets into which the uncertainty about the initial state can be partitioned, state-identification algorithms are only concerned with the decision problem, that is whether or not a full identification, i.e, a partitioning into singleton knowledge sets is feasible, and if it is, by which strategy. So our problem of finding the finest partition can be seen as a quantitative generalization of the state-identification problem.

A proposal to quantify the security of cache memories was introduced in [26]. In this case, they use several types of attackers and study the security under different countermeasures, without considering the replacement policies individually. They obtained arguments in favor of some countermeasures against specific attacks. In our case we consider one single type of attacker, do not take into account any type of countermeasure and compare the different replacement policies.



(a) Filled cache using FIFO.

(b) Empty cache using FIFO.

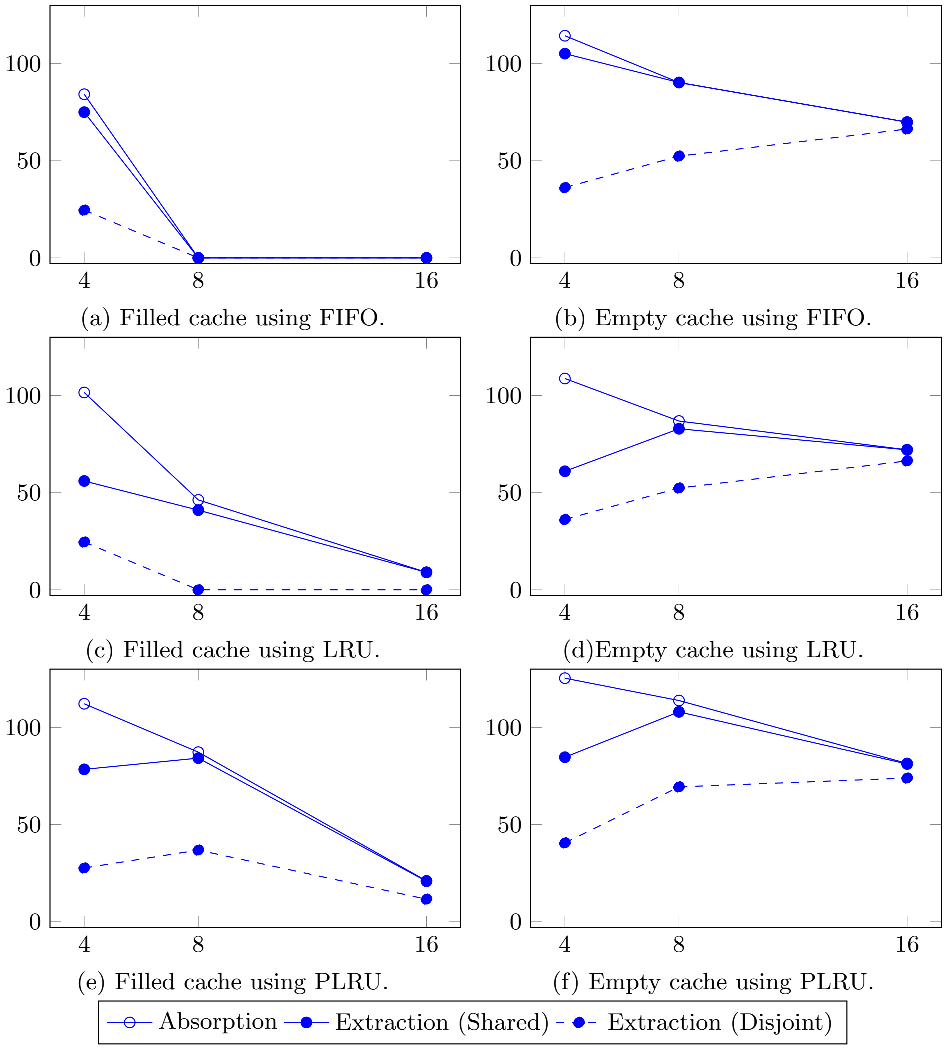
(c) Filled cache using LRU.

(d) Empty cache using LRU.

(e) Filled cache using PLRU.

(f) Empty cache using PLRU.

Fig. 3: Information extraction of different replacement policies on a 4-way cache set. Figures 3a, 3c and 3e depict the case of a filled initial cache, 3b, 3d and 3f an empty one. In all figures, the horizontal axis depicts the footprint, i.e., the number of memory blocks used. The vertical axis depicts the extracted information on a logarithmic scale, that is, in *bits*. The results for shared memory adversaries use the solid line; disjoint memory case uses the dashed line.



(a) Filled cache using FIFO.

(b) Empty cache using FIFO.

(c) Filled cache using LRU.

(d)Empty cache using LRU.

(e) Filled cache using PLRU.

(f) Empty cache using PLRU.

Fig. 4: Information absorption and extraction (in bits) for the AES execution on a 4-way cache, for filled and empty initial cache states. Figures 4a, 4c and 4e depict the case of a filled initial cache, 4b, 4d and 4f an empty one. The horizontal axis depicts the size of the cache in KB, the vertical axis depicts the extracted information in logarithmic scale.

8 Future Work and Conclusions

We presented a novel approach for quantifying isolation properties of shared caches, based on a simple model of adaptive attacks against Mealy machines. We use our approach for performing the first security analysis of common cache replacement policies (LRU, FIFO, PLRU), as well as for improving the precision of the CacheAudit static analyzer. Our prime target for future work is to investigate an extension of our model to Markov Decision Processes for dealing with randomized replacement policies.

Acknowledgments We thank Pierre Ganty and the anonymous reviewers for their constructive feedback.

This work was supported by Microsoft Research through its PhD Scholarship Programme, by Ramón y Cajal grant RYC-2014-16766, Spanish projects TIN2012-39391-C04-01 StrongSoft and TIN2015-70713-R DEDETIS, and Madrid regional project S2013/ICE-2731 N-GREENS, and by the German Research Council (DFG) as part of the Project PEP.

References

1. A. Abel and J. Reineke. Measurement-based modeling of the cache replacement policy. In *RTAS*, pages 65–74. IEEE, 2013.
2. O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *ASIACCS*, pages 312–320. ACM, 2007.
3. O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. Predicting secret keys via branch prediction. In *CT-RSA*, pages 225–242. Springer, 2007.
4. M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *CSF*, pages 265–279. IEEE, 2012.
5. A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *SSP*, pages 207–221. IEEE, 2007.
6. D. Bernstein. Cache-timing attacks on AES. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.
7. M. Boreale and F. Pampaloni. Quantitative multirun security under active adversaries. In *QEST*. IEEE, 2012.
8. P. Cañones, B. Köpf, and J. Reineke. Security analysis of cache replacement policies. `http://arxiv.org/abs/1701.06481`, 2017.
9. P. J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, 1968.
10. G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke. CacheAudit: a tool for the static analysis of cache side channels. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):4:1–4:32, 2015.
11. A. Gill. State-identification experiments in finite automata. *Information and Control*, 4(2-3):132–154, 1961.
12. D. Gullasch, E. Bangerter, and S. Krenn. Cache games - bringing access-based cache attacks on AES to practice. In *SSP*, pages 490–505. IEEE, 2011.
13. B. Köpf and D. Basin. An Information-Theoretic Model for Adaptive Side-Channel Attacks. In *CCS*, pages 286–296. ACM, 2007.

14. B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.
15. D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on computers*, 43(3):306–320, 1994.
16. F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *SSP*, pages 605–622. IEEE, 2015.
17. P. Mardziel, M. S. Alvim, M. Hicks, and M. R. Clarkson. Quantifying information flow for dynamic secrets. In *SSP*, pages 540–555. IEEE, 2014.
18. E. F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.
19. M. Neve and J. Seifert. Advances on access-driven cache attacks on AES. In *SAC*, pages 147–162. Springer, 2006.
20. D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of AES. In *CT-RSA*, pages 1–20. Springer, 2006.
21. G. Smith. On the foundations of quantitative information flow. In *FoSSaCS*, pages 288–302. Springer, 2009.
22. M. Tiwari, J. Oberg, X. Li, J. Valamehr, T. E. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood. Crafting a usable microkernel, processor, and I/O system with strict and provable information flow security. In *ISCA*, pages 189–200. ACM, 2011.
23. X. Xiang, C. Ding, H. Luo, and B. Bao. HOTL: a higher order theory of locality. In *ASPLOS*, pages 343–356. ACM, 2013.
24. Y. Yarom and K. Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX*, pages 719–732. USENIX Association, 2014.
25. D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers. A hardware design language for timing-sensitive information-flow security. In *ASPLOS*, pages 503–516. ACM, 2015.
26. T. Zhang and R. B. Lee. New models of cache architectures characterizing information leakage from cache side channels. In *ACSAC*, pages 96–105. ACM, 2014.