A Predictive Model for Cache-Based Side Channels in Multicore and Multithreaded Microprocessors

Leonid Domnitser, Nael Abu-Ghazaleh, and Dmitry Ponomarev

Computer Science Department
State University of New York at Binghamton
Binghamton, NY 13902
{lenny,nael,dima}@cs.binghamton.edu

Abstract. A side channel is an information channel that unintentionally communicates information about a program as a side effect of the implementation. Recent studies have illustrated the use of shared caches as side channels to extract private keys from computationally secure cryptographic applications. The cache side channel is imperfect in the sense that the attacker's ability to detect cache leakage of critical data is limited by the timing issues. Moreover, some detected leakages are due to non-critical data. Thus, it is difficult to assess the degree of vulnerability given the imperfect nature of the side-channel. Similarly, when solutions that further degrade the quality of the channel, but do not necessarily close it completely, are employed, it is difficult to evaluate their effectiveness. To address this need, this paper proposes a mathematical model to evaluate the expected leakage in a cache as a function of the cache parameters and the victim application behavior. We use simulation to quantify these parameters for typical attack scenarios to validate the model. We demonstrate that the proposed model accurately estimates side channel leakage for for AES and Blowfish encryption and decryption on a variety of cache configurations.

Keywords: architecture, security, side channel attack, caches.

1 Introduction

In recent years, security has emerged as one of the key design issues in computing and communication systems. Security solutions typically rely on a set of cryptographic algorithms, such as symmetric ciphers, public-key ciphers, and hash functions. The strength of modern cryptography makes it infeasible for the attackers to uncover the secret keys used in these algorithms by brute-force trials, differential [9] or linear cryptanalysis [14]. Instead, almost all known attacks on the secret keys today exploit weaknesses in the physical implementation of the system performing the encryption, rather than exploiting the mathematical properties of the cryptographic algorithm itself.

A subtle form of vulnerability in the physical implementation of otherwise secure systems is a possible leakage of information through unintended (or side) channels. The leaked information is called *side-channel information*, and the attacks exploiting side-channel information leakage are called *side-channel attacks* [1,19,21]. Examples of side-channels include observation of execution time, power consumption, heat, electromagnetic radiation, or even sound emanating from a device [21]. A large number

I. Kotenko and V. Skormin (Eds.): MMM-ACNS 2010, LNCS 6258, pp. 70–85, 2010.

of side-channel attacks have been successfully demonstrated against a range of software and hardware security mechanisms: they have been used to break many cryptosystems including block ciphers (such as DES, AES, Camellia, IDEA, and Misty1), stream ciphers (such as RC4, RC6, A5/1, and SOBER-t32), public key ciphers (such as RSA-type ciphers, ElGamal-type ciphers, ECC, and XTR), signature schemes, message authentication code schemes, cryptographic protocols, and even the networking subsystems[21]. Thus, it is critical to build systems that are immune to side-channel attacks.

Traditionally, side-channel attacks were used to break simple systems such as smart cards. However, a new class of attacks that exploit the shared caches in microprocessors as side-channels had recently emerged as a serious security threat [26,27,12,24,18,19,1,3,5]. The nature of this new threat is rooted in the ability of modern microprocessors to execute several programs concurrently on the same chip to exploit so-called Thread-Level Parallelism (TLP). TLP is exploited by the processor designers in two ways: Simultaneous Multithreading (SMT) and Chip Multiprocessing (CMP, also called multicore). In an SMT processor [25], several independent programs are simultaneously executing on the same processing core, and most of the core's resources, including the on-chip caches, are shared among the threads. In contrast, CMPs consist of completely replicated processing cores that share only a small subset of resources such as lower level caches, main memory and I/O pins.

Consider the concurrent execution of two programs in an SMT environment – a security-critical encryption kernel (which we refer to as the "victim") and a process performing an attack on the secret key used by the victim (which we refer to as the "attacker"). By sharing the data cache with the victim, the attacker can detect the victim's cache accesses when the victim evicts the data belonging to the attacker. The detection is possible because on its next access to the same data, the attacker will miss into the cache, and cache misses can be easily distinguished from hits by using timing instructions readily available in most modern innstruction sets. Several studies [19,1,3,24] showed that the information leaked through the cache side channel is often sufficient to reconstruct the full secret key in a short period of time. Similar leakage is possible through the shared lower-level caches in multicore system, even without multithreading in individual cores.

Although cache-based side-channel attacks have been demonstrated, a successful attack involves gleaning of the critical information from an imperfect channel. In particular, some memory accesses may not be leaked at all (we explain the reasons for that in detail in the later sections). Moreover, some non-critical accesses may be detected; these accesses do not correlate with the secret key and therefore add noise to the information collected from the side-channel. Thus, key reconstruction involves a significant effort, depending on the amount and quality of the information detected from the side-channel. If the amount of useful information collected through the side channel is small, key reconstruction may require prohibitive computational overhead, or just fail.

Being able to quantify this relationship between a side channel leakage properties and the computational difficulty of compromise is critical. It allows quantitative evidence of vulnerabilities. Moreover, it may be impossible or extremely expensive to

completely shut down the side-channel[26,27]. Thus, low complexity solutions that reduce the quality of the channel may be of interest. However, it is difficult to accept such defenses based on informal evidence; being able to formally quantify the security of imperfect channel can lead to effective solutions that have acceptable complexity while providing sufficient security.

To help the computer designers implement the right level of protection against cache-based attacks, two key questions have to be addressed: 1) How much information is leaked through the side channel, and 2) what is the effort required to convert this information into a full secret encryption key. In this paper, we address the first question and develop a simple analytical model to predict the amount of critical information leakage through the cache-based side channel. The model takes into account the capabilities of the attacker, the parameters of the victim process and the hardware configuration of the cache. We validate the developed model through cycle-accurate simulations of two encryption kernels (AES and Blowfish) and demonstrate that the side channel leakege predicted by the analytical model matches simulation-based results. Finally, we explain how the designers can use the proposed model to reason about the threat level, the impact of different attack optimizations, and the impact of possible defense approaches.

The remainder of the paper is organized as follows. We review the AES and Blowfish algorithms and analyze why they are vulnerable to cache-based attacks in Section 2. Sesction 3 describes the proposed leakage prediction model. In Section 4, we present the simulation methodology and simulation results to validate the model. Section 5 reviews the related work and we conclude in Section 6.

2 Background

In this section we describe how the Advanced Encryption Standard (AES) and Blowfish encryption lend themselves to exploitation by side channel attackers. We also explain how a side channel attack through the L1 data cache works. An attack on last level cache in a multicore system can be performed in a similar fashion.

2.1 The Advanced Encryption Standard (AES)

AES, the Advanced Encryption Standard, is a widely used symmetric block cipher. It encrypts and decrypts 128-bit data blocks using either a 128-, 192-, or 256-bit key. Each block is encrypted in 10 rounds of mathematical transformations. To achieve high performance, AES implementations use precomputed lookup tables instead of computing the entire transformation during each round. The indexes to these tables are partially derived from the secret key, thus by detecting the cache sets accessed by the victim (through the side channel observations), the attacker can derive some information about parts of the secret key. By using multiple measurements, the entire key can be successfully reconstructed. The version of the AES code that we use in this study [6] employs five tables (1KB each) for both encryption and decryption. The first four tables are used in the first nine rounds of encryption/decryption, and the fifth table is used during the

last round. Separate sets of tables are used for encryption and decryption. More details on the AES encryption algorithm and specific side channel attacks on AES can be found in [24].

2.2 The Blowfish Encryption Algorithm

Blowfish [2] is a keyed, symmetric block cipher, included in a large number of cipher suites and encryption products. Blowfish has a 64-bit block size and a variable key length from 32 up to 448 bits. It is a 16-round Feistel cipher and uses large key-dependent S-boxes. The algorithm keeps two subkey arrays: the 18-entry P-array and four 256-entry S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries. The F-function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output. Again, just in the case with AES, the accesses to S-boxes are the critical accesses that can reveal the key-related information through the cache side channel.

2.3 Cache-Based Side Channel

A dangerous side channel exists if an attacker can determine which table rows, which we call *critical data*, are accessed. A shared memory cache can carry such a side channel. The time to access data present in the cache (a cache hit) is different from the time to access data not in the cache (a cache miss), so it is possible to tell which data is in the cache by measuring access time.

Caches typically have a set-associative organization. Each memory location is part of a multi-byte data block called a cache line, and several lines are grouped into a cache set. When data is loaded, an entire line is brought in and is deterministically mapped, by address, into a specific set. Multiple lines coexist in a set (the number of lines in a set is the associativity of the cache), but when new lines are loaded, an old line must usually be evicted. This is done according to a replacement policy, such as evicting the least recently used (LRU) line.

When a line of critical data is loaded, it replaces some other line. If the cache is shared with an attacker, the evicted line may belong to the attacker. By later accessing that line, and timing that access, the attacker learns whether the victim accessed an address mapping to the same set.

The Attack. Given such a side channel, an attack is relatively straightforward. An attacker fills the entire cache, ensuring that any memory access by the victim will evict the attacker's data. After a cryptographic operation, the attacker returns to each cache set and accesses the same data to determine if it misses, indicating that the victim accessed the set.

The attacker needs to perform the following steps:

- Gain user-level access to the computer performing cryptography.

- It must also be aware of the cache configuration—number of sets, associativity, line size, replacement policy, etc. This might be known ahead of time, or it might be programatically deduced.
- For a side channel to be useful, the attacker must be somehow synchronized with the victim. For simplicity, we assume that a synchronous attack is possible. That is, the attacker can trigger the cryptographic process, say, by an inter-process communication or networking mechanism. If an attacker can trigger single–data-block encryption or decryption, it need not worry about keeping pace with the victim to ensure that the cache stays full of attacker data.
- The attacker cannot "look within" a line. When an attacker line is evicted, all it can learn is that the victim accessed a location within a particular line, not the specific critical data index within the line. This is the nature of the side channel—an attacker will have to do some brute force work.
- The critical data accessed can be difficult to determine, not just because of line-size granularity, but because other, unrelated victim data can map to the same set, creating noise in the side channel.

3 Model for Side Channel Leakage Prediction

The goal of the proposed model is to predict the probability that a critical cache access (that is, the access to the critical data that is dependent on the secret key) is exposed on a cache-based side channel, both in aggregate and per each cache set. More precisely, we predict the conditional probability that the attacker detects a memory access on the side channel, given that there was access to critical data. This probability can be expressed as $P(D|C)$, where D is the event of detection, and C is the event of a critical access. $P(D|C)$ is defined to be equal to $\frac{P(D \cap C)}{P(C)}$. Using a few basic algebraic transformations of this definition, we obtain a simple statement of Bayes' theorem, which gives the relationship between a conditional probability and its inverse:

$$P(D|C) = \frac{P(C|D)\,P(D)}{P(C)} \tag{1}$$

Table 1. Summary of symbols

A	event of an access
α	number of memory accesses
C	event of critical access
D	event of detected access
m	number of lines used in a set
N	number of sets
s	cache set number
T_a	time between repeat accesses by attacker
T_v	time between repeat accesses by victim
w	cache associativity

This formula is the basis of the proposed model. We predict the variables $P(C|D)$ (the conditional probability that there was a critical access, given that the attacker detected an access), $P(D)$, and $P(C)$ based on properties of the attacker and victim programs, and the system on which they execute. In the next section, we measure the variables through simulation and evaluate our predictions.

$P(D)$ and $P(C)$ refer to the probabilities of events D and C *when an access occurs*, not for any instruction. That is, we assume an initial condition A, that there is a memory access. So $P(D)$ is a shortened notation for $P(D|A)$. Accordingly, $P(D)$ and $P(C)$ are affected by the real timeline of CPU cycles, but are calculated relative to the timeline of memory access instructions only.

The events A, C, and D can refer to all accesses, or can be restricted accesses to a subset of data. We consider both aggregate measures and those restricted to individual cache sets. In the latter case, A_s, C_s, and D_s refer to the events of accesses to set s.

The rest of this section analyzes the components variables of the above Bayesian formula, then combines the variable predictions into a single formula to predict side channel exposure of critical data.

3.1 Estimating Critical Accesses ($P(C)$)

We define the probability of a critical access, $P(C)$ as the average rate of critical accesses. This probability defines how many accesses during the execution of a cryptographic program are critical, if the total number of accesses is α. This value is an invariant property of the implementation of a cryptographic algorithm, and can be estimated through static analysis or profiling.

In our Bayesian prediction formula, $\frac{1}{P(C)}$ is a constant for a given program. However, it varies among cryptographic algorithms (and implementations), and between encryption and decryption routines.

3.2 Access Detection ($P(D)$)

The probability that the attacker detects an access, $P(D)$, is the number of detected accesses out of the total number of accesses.

$$P(D) = \frac{\alpha_D}{\alpha} \tag{2}$$

$P(D)$ is 100% for a perfect attacker, which measures the victim without error after every instruction. Realistically, there are several reasons that a memory access may be hidden from the side channel. The number of hidden accesses is $\alpha_{\neg D}$, and $P(D)$ can be restated in terms of hidden accesses:

$$P(D) = 1 - \frac{\alpha_{\neg D}}{\alpha} \tag{3}$$

The simplest reason that a memory access may not be detected is a cache hit. The attacker only sees a victim's access if it misses into the cache and evicts attacker's data. The attacker's data is not automatically placed back in the cache when the victim evicts it; rather, the attacker scans and refills the particular cache location at some point after

the victim accesses it. In the time it takes the attacker to traverse the cache, the victim may have performed several accesses to the same cache location, and all but one of those will remain undetected by the attacker.

The percentage of accesses hidden from the side channel by the cache hits depends on the victim's pattern of access to individual critical data entries and the speed with which an attacker returns to each set. These behaviors can vary depending on victim and attacker implementations, as well as the cache configuration.

For a fixed cache configuration, the victim's "rate of return" to a particular critical data entry depends on the input data and the secret key. The average rate is a property of the cryptographic code. Since the side channel operates at the granularity of cache lines, the cache line size also affects the hit rate. The victim does not necessarily have to access the exact same critical data to hit into the cache, but must access data within a resident cache line. Larger cache lines, besides confounding multiple possible critical data addresses when an access *is* detected, increase the likelihood of cache hits which are hidden from the side channel.

The attacker can also be accelerated by larger cache lines, since fewer memory accesses are required to scan the entire cache. If the rate of return for both the victim and attacker scale linearly with the line size, then the effect is canceled out. However, a victim automatically exploits the increased hit rate, while an attacker may have to address synchronization issues that result. In a synchronous attack that is synchronized at block encryption boundaries, increased line size helps only the victim. Even in a purely asynchronous attack, where "synchronization" is done by post-attack analysis, the attacker must still emit or save its timing results after each cache traversal, which will happen more often with shorter traversal times.

Depending on the attack code, the cache dimensions—number of sets and associativity—can affect the speed of the attack. At the cost of higher code complexity, an attacker can reduce the number of accesses it must perform by accessing each set just once[1]. If this sort of an attack is used, then the attacker can traverse a highly associative cache with fewer sets faster than it can traverse a less associative cache (with more sets) of the same size. Similar to increasing line size, increasing associativity can increase the attack speed and decrease hidden critical accesses, but only if this optimized attack is used, and if synchronization issues are handled.

Memory accesses can also be hidden from the side channel by design. Hardware or software defense mechanisms can reduce $P(D)$. A perfect defense mechanism reduces $P(D|C)$ to 0. Of course, if no critical accesses are detected, the side channel does not exist. Specific defense mechanisms are outside the scope of this paper, but they can be captured in this portion of the model.

3.3 Estimating Critical Accesses Given Detection ($P(C|D)$)

$P(C|D)$ is the fraction of detected accesses that are critical. This represents how clean the signal on the side channel is—100% means that every access that is detected is useful to the attacker. A real side channel, however, has sources of noise. Noise is expressed

[1] Assuming the cache replacement policy is Least Recently Used (LRU), the attacker can ensure that it always accesses the LRU line of a set. If that is a hit, the other lines in the set will also hit.

as $P(\neg C|D)$, or the probability that there is no critical access, given a detected access. (With our assumptions, C also means that there was a non-critical access.) Since we consider the cause of noise, not of signal, we represent $P(C|D)$ as $1 - P(\neg C|D)$.

Even with no complicating factors, some noise is inherent in the cryptographic implementation. A victim's access to non-critical data that maps to the same cache set as critical data cannot be distinguished from a critical access. The attacker, in its analysis stage, can try to filter out a pattern of noise. An attack must have tolerance for noise, so that some brute force trials can be performed while still utilizing data from the side channel as a hint.

Noise can also come from the attacker's side, in the form of instruction cache misses, from misses in the translation lookaside buffer, from the operating system, or from anything else that may confound timing results.

3.4 Model Formalization

We now unify the ideas presented above into a formal predictive model. We return to our Bayesian formula, predicting side channel exposure in a single set:

$$P(D_s|C_s) = \frac{P(C_s|D_s)\,P(D_s)}{P(C_s)} \tag{4}$$

The probability of a critical access is simply the ratio of critical accesses to total accesses:

$$P(C_s) = \frac{\alpha_{C,s}}{\alpha_s} \tag{5}$$

To model the probability of a detected access, we first consider the average likelihood that an access will be hidden by hitting into the cache. Repeated victim's access to a cache location before the attacker scans and refills that location will be hidden from the side channel. Consequently, in the time it takes an attacker to return to a location, T_a, all accesses after the first one will be hidden. The time between repeated accesses to that location is T_v. The probability of an access to a single location being detected, then, is $\frac{T_v}{T_a}$, assuming there is an intervening victim access to the location ($T_v \leq T_a$). T_v, on average across accesses during an attacker traversal, is expressed as $\frac{T_a}{\alpha_{location}}$. So we obtain:

$$P(D_{location}) = \frac{1}{\alpha_{location}} \tag{6}$$

Since we assume that a cache access has taken place, $\alpha > 0$, this value is defined.

Next, we expand our model to a cache set, rather than a single location. There is some number of lines of data mapping to set s, which we call m_s. The cache is w-way associative. If $m_s \leq w$, then all the data fits in the set without eviction, and accesses to all these data can be detected. In this case $P(D_s) = \frac{m_s}{\alpha_s}$. When $m_s > w$, only $\frac{w}{m_s}$ of the data lines used are resident in the cache, and can possibly leak information. This is a "fit factor" that limits the probability of detection in a set. Generalizing, we obtain:

$$P(D_s) = \frac{m_s \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_s} \tag{7}$$

$m_s \cdot \min\left(\frac{w}{m_s}, 1\right)$ can be thought of as the "pressure" on the cache set.

We now model the probability of critical access, given detected access, which can be expressed as $1 - noise$. The simple noise we consider is caused by non-critical accesses. This model considers the number of critical accesses to a set $\alpha_{C,s}$, which along with non-critical accesses $\alpha_{\neg C,s}$, make up all accesses α_s. We start with the definition of conditional probability:

$$P(C_s|D_s) = \frac{P(C_s \cap D_s)}{P(D_s)} \tag{8}$$

$P(D_s)$ is already modeled. The model for $P(C_s \cap D_s)$ is very similar—we just look at the subset of detected accesses that are critical. $m_{C,s} \leq m_s$ is the number of critical data lines mapping to set s. We restrict $P(D)$ to just these lines:

$$P(C_s \cap D_s) = \frac{m_{C,s} \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_s} \tag{9}$$

Thus, the equation for $1 - noise$ is the following:

$$P(C_s|D_s) = \frac{\left(\frac{m_{C,s} \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_s}\right)}{\left(\frac{m_s \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_s}\right)} = \frac{m_{C,s}}{m_s} \tag{10}$$

We can now unify the model into a single formula:

$$P(D_s|C_s) = \frac{\left(\frac{m_{C,s}}{m_s}\right)\left(\frac{m_s \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_s}\right)}{\left(\frac{\alpha_{C,s}}{\alpha_s}\right)} \tag{11}$$

Our single-set model simplifies to:

$$P(D_s|C_s) = \frac{m_{C,s} \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_{C,s}} \tag{12}$$

Finally, we consider the entire cache rather than a single set. We ignore sets without critical accesses, since the condition of the probability is that there were critical accesses, and take the average:

$$P(D|C) = \frac{\sum_{s=0}^{N} \begin{cases} 0 & : \alpha_{C,s} = 0 \\ \frac{m_{C,s} \cdot \min\left(\frac{w}{m_s}, 1\right)}{\alpha_{C,s}} & : otherwise \end{cases}}{\sum_{s=0}^{N} \begin{cases} 0 : \alpha_{C,s} = 0 \\ 1 : otherwise \end{cases}} \tag{13}$$

This model predicts the effectiveness of a side channel using only a simple profile of the victim (distribution of critical and overall cache accesses within a period of attack), and cache associativity.

4 Model Validation Methodology

To validate the model proposed in the previous section, we performed cycle-accurate simulations of AES and Blowfish encryption algorithms running alongside the idealized attacker on an SMT processor. We used M-Sim 3.0 [17]—a SMT and CMP simulator that was derived from Simplescalar 3.0d [7]. The crypto programs were compiled on an Alpha AXP machine running Tru64 UNIX, using the native C compiler with `-04 -fast -non_shared` optimization flags. We simulated an 8-way processor with 128-entry Reorder Buffer and a 32KB L1 data cache under several configurations.

We simulated both an 8-way set-associative cache, and a direct-mapped (1-way) cache. The 8-way cache is a typical configuration, while the direct-mapped cache was used to exercise the fit factor of the model. Both caches used 32 byte lines, so the caches had 128 and 1024 sets, for 8-way and direct-mapped, respectively.

We simulated the execution of an idealized attacker as a separate thread alongside an encryption or decryption process. It is "idealized" because it is implemented with simulator support to synchronize perfectly with cryptographic block operations. The only noise, therefore, occurred during the core cryptographic operation, and the attacker did not operate under time constraints. This is the best an attacker can do, and a is worst-case bound on security.

We ran simulations of 1,000 truly random [20] input data blocks. The experiments included all permutations of the following operations:

- Algorithm: AES or Blowfish
- Operation: encryption or decryption
- Cache configuration: 8-way or direct-mapped

Our simulations outputted memory access traces; a script used these data to generate predictions. We also generated attacker traces which labeled memory accesses as detected or not. For each block operation (that is, each instance of the attack), the model matches the measured detection rates. This makes sense, because the same data block is used both to profile the victim for the model and to measure the side channel. We also show how the average side channel that we measure as a profile is a good predictor for individual blocks. Our experiments show low variation in side channel leakage from block to block.

5 Results and Evaluation

Figure 1 shows the aggregate detection rate averaged across 1,000 cryptographic block operations. This is the measured rate of detection, which matches the rate predicted by

our model for each of the same blocks. The average value of $P(D|C)$ is 76% for AES and 87% for Blowfish on the 8-way cache. The direct-mapped cache exhibits similar results, 77% and 88% for AES and Blowfish, respectively. Figure 2 shows same side channel leakage broken down by set. The 8-way cache has a fairly consistent distribution of accesses among its 128 sets, since the critical data tables are spread evenly across all sets. The direct-mapped cache only shows leakage in 256 sets for AES and 268 sets for Blowfish, out of 1024 available sets. This is because only those sets have critical data (since direct mapped cache has a larger number of sets).

Figure 3 shows, for each experiment, how the detection rate for each block compares to the average rate across all the blocks. These graphs show bell curves in a small range around the average, with a standard deviation of 2% in all cases. The maximum deviation is 7% for AES and 8% for Blowfish. Therefore, our simple predictive model which is based on the average detection rate across all blocks predicts the detection within individual blocks with an error of only 8% in the worst case.

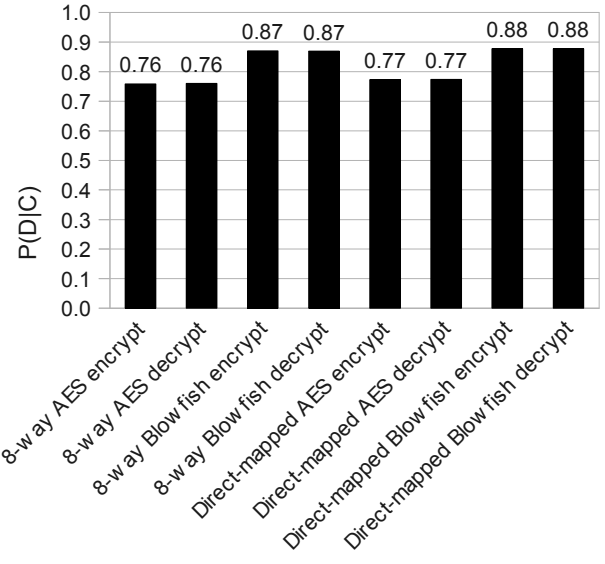
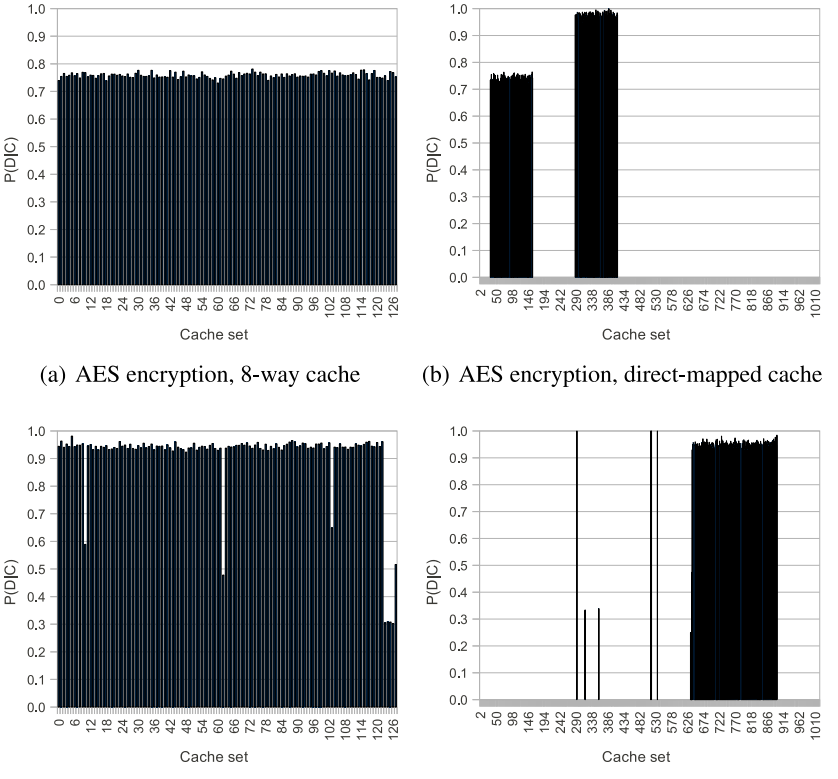


Fig. 1. Aggregate detection rate. This graph shows the detection rate ($P(D|C)$) as an average across 1,000–data block cryptographic operations. Predicted and measured detection rates are equal for each block operation.

Therefore, our experiments confirm that the results obtained by the proposed model closely match the simulated results. Furthermore, they demonstrate that different input data causes only a low level of variation in side channel leakage, so simple metrics, such as estimated average characteristics across all blocks can be used for accurately predicting the amount of leaked data on a block-by-block basis.



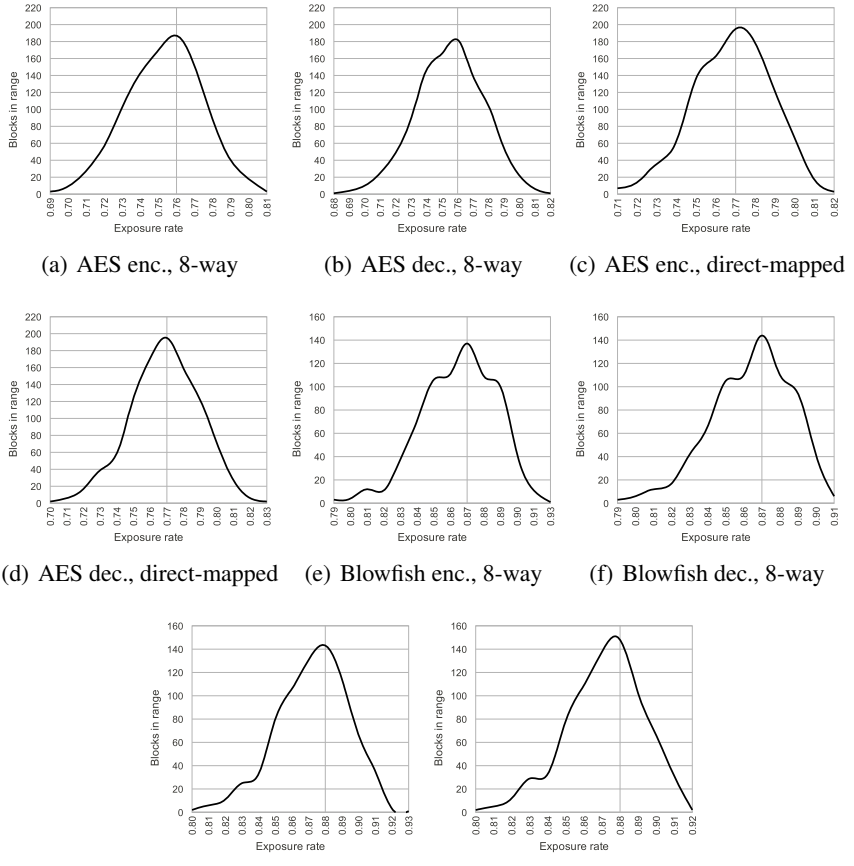
(a) AES encryption, 8-way cache

(b) AES encryption, direct-mapped cache

(c) Blowfish encryption, 8-way cache

(d) Blowfish encryption, direct-mapped cache

Fig. 2. Detection by set. These graphs show detection rate in each set ($P(D_s|C_s)$) as averages across 1,000–data block encryptions (decryption graphs are omitted because they are nearly identical). Predicted and measured detection rates are equal for each block operation.



(a) AES enc., 8-way (b) AES dec., 8-way (c) AES enc., direct-mapped

(d) AES dec., direct-mapped (e) Blowfish enc., 8-way (f) Blowfish dec., 8-way

(g) Blowfish enc., direct-mapped (h) Blowfish dec., direct-mapped

Fig. 3. Each graph shows the number of block operations with side channel leakage within a given 1% range

6 Related Work

The security of cryptographic implementations with respect to side-channel attacks has not been widely investigated. Despite the presence of a number of a solutions to side-channel problems that do not perfectly close the channel [10,15], the security properties of side-channels and the effectiveness of such imperfect solutions were open questions. Micali and Reyzin were the first to present a theoretical analysis of general side-channel attacks [16]. Using very general assumptions, this model defines the notion of an abstract computer and a leakage function that together can capture almost all instances of side channels. However, the overly general assumptions make it difficult to apply this analysis to particular algorithms (e.g., DES or AES) or for specific side-channels.

Standaert *et al.* started from Micali and Reyzin model and specialized it for more practical situations [23]. Specifically, they restricted some of the assumptions to a range that corresponds to relevant adversary and leakage models. Moreover, they show how to map the abstract computational model to physical instances such as circuits and operations. Although this model brings the original model by Micali and Reyzin closer to practice, it models the leakage and adversary abstractly using information theoretic principles. More recently, Standaert *et al.* created a uniform model of side-channel attacks to address the problem of how compare different algorithm implementations and defense mechanisms in a way that enables comparing them [22].

Kopf and Basin developed an information theoretic model of side-channels [13]. Like our model, they restrict their analysis to the amount of information leaked from the channel. This model considers a generic side channel, and does not capture the detailed operation of cache based side-channel attacks that we characterize in this paper.

Both software and hardware solutions to address cache-based side channel attacks have been proposed. On the software side, the main idea is to rewrite the code of the encryption algorithms such that known side channel attacks are not successful. Examples of such techniques include avoiding the use of table lookups in AES implementations, preloading the AES tables into the cache before the algorithm starts, or changing the table access patterns [18,24,4,21]. The limitation of the software solutions is that that they are tied up to a specific algorithm/attack, do not provide protection in all cases, are subject to errors on the part of programmers, and often result in significant performance degradation [26]. Another recent approach to address side channel attack is by dedicating special functional units and ISA instructions to support a particular cryptographic algorithm. An example of this approach is the Intel AES instruction [11]. This, however, requires non-trivial hardware and software changes and only protects against the attacks on the crypto algorithms that are supported—support has to be re-implemented to defend new algorithms.

In response to the limitations of software solutions, several hardware schemes have been recently introduced. The advantage of hardware solutions is that they prevent the attacks in principle, by eliminating the side channel. The main challenge in these schemes is to keep the impact on the design complexity, cache access time, and performance overhead to the minimum. Following this line of research, a partitioned cache was proposed [8], along with ISA changes to make the cache a visible part of the architecture. Specifically, new instructions are added to define a partition and specify its size and parameters. This scheme requires changes to both the ISA and the cache

hardware design and can lead to significant performance degradation. Several alternative cache designs for thwarting cache-based attacks have been proposed by Wang and Lee [26,27]. Partition-Locked Cache (PL cache) design [26] uses cache line locking to prevent evictions of cache lines containing critical data, thus closing the side channel. The main drawback is the performance hit due to cache underutilization, as the locked lines cannot be used by other processes, even after they are no longer needed by the process that owns them. In addition, the PLcache requires system support to control which cache lines should be locked. This support is in the form of new ISA instructions, or OS modifications for marking the regions of memory that contain the AES or RSA tables as lockable. In either case, ISA/compiler/OS modifications are also needed in addition to the hardware changes.

7 Conclusion

Cache-based software side-channel attacks represent a new and serious security threat that exploits parallel processing capabilities of modern processor chips. Defense mechanisms that provide a complete closing of the side channel are expensive and often incur significant performance overhead. A possible alternative is to consider solutions that do not result in a complete elimination of the side channel, but rather attempt to reduce its strength to the levels that make the remaining post-attack effort for the secret key reconstruction infeasible.

To assist system designers with such solutions, we developed an analytical model for estimating the percentage of accesses to the critical data that would be leaked through the cache side channel as a function of the victim's characteristics and the configuration of the cache hardware. We validated the proposed model using cycle-accurate simulation of side-channel attack on two popular encryption kernels (AES and Blowfish) and also described how the model can be used in exploring the design space of low-complexity solutions for cache-based attacks.

Acknowledgements. This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-09-1-0137. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of Air Forse Research Laboratory or the U.S. Government.

References

1. Bernstein, D.: Cache-timing attacks on aes (2005), `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`
2. The blowfish encryption algorithm (2009), `http://www.schneier.com/blowfish.html`
3. Bonneau, J., Mironov, I.: Cache-collision timing attacks against aes. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 201–215. Springer, Heidelberg (2006)

4. Brickell, E., Graunke, G., Neve, M., Seifert, J.: Software mitigation to hedge aes against cache-based software side channel vulnerabilities. In: IACR ePrint Archive, Report 2006/052 (2006)
5. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding cache attacks. INRIA Technical Report (2006), ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-5881.pdf
6. Daemen, J., Rijmen, V.: The design of rijndael: Aes - the advanced encryption standard. Springer, Heidelberg (2002)
7. Burger, D., Austin, T.: The simplescalar toolset: Version 2.0 (June 1997)
8. Page, D.: Partitioned cache architecture as a side-channel defense mechanism. In: Cryptography ePrint Archive (2005)
9. Biham, E., Shamir, A.: Packaging of multi-core microprocessors: Tradeoffs and potential solutions. Journal of Cryptology 4(1), 3–72 (1991)
10. Goubin, L., Patarin, J.: DES and differential power analysis. In: Proc. of CHES (1999)
11. Gueron, S.: Advanced encryption standard (aes) instruction set (2008)
12. Kong, J., Aclicmez, O., Seifert, J., Zhou, H.: Hardware-software integrated approaches to defend against software cache-based side channel attacks. In: International Symposium on High Performance Computer Architecture (HPCA) (February 2009)
13. Kopf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: ACM Conference on Computer and Communication Security (CCS), pp. 286–296 (2007)
14. Matsui, M.: Linear cryptanalysis method for des cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
15. May, D., Muller, H., Smart, N.: Randomized register renaming to foil DPA. In: Proc. of CHES (2001)
16. Micali, S., Reyzin, L.: Physically observable cryptography. In: Proc. of Theory of Cryptography Conference (2004)
17. M-sim version 3.0, code and documentation (2005), http://www.cs.binghamton.edu/~msim
18. Osvik, D., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of aes. In: Cryptology ePrint Archive, Report 2005/271 (2005)
19. Percival, C.: Cache missing for fun and profit (2005), http://www.daemonology.net/papers/htt.pdf
20. Random.org (2009), http://www.random.org/
21. Side channel attacks database (2009), http://www.sidechannelattacks.com
22. Standaert, F.X., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Advances in Cryptography, Eurocrypt (2009)
23. Standaert, F.X., Peeters, E., Archambeau, C., Quisquater, J.J.: Towards security limits in side-channel attacks. In: Proc. CHES Workshop (2006)
24. Tromer, E., Shamir, A., Osvik, D.: Efficient cache attacks on aes, and countermeasures. Journal of Cryptology (2009)
25. Tullsen, D., Eggers, S., Levy, H.: Simultaneous multithreading: Maximizing on-chip parallelism. In: International Symposium on Computer Architecture (1995)
26. Wang, Z., Lee, R.: New cache designs for thwarting software cache-based side channel attacks. In: Proc. International Symposium on Computer Architecture (ISCA) (June 2007)
27. Wang, Z., Lee, R.: A novel cache architecture with enhanced performance and security. In: Proc. International Symposium on Microarchitecture (MICRO) (December 2008)