A Comparison Study on Flush+Reload and Prime+Probe Attacks on AES Using Machine Learning Approaches

Zirak Allaf, Mo Adda, and Alexander Gegov

University of Portsmouth, Portsmouth, Hampshire, UK,
{zirak.allaf,mo.adda,alexander.gegov}@port.ac.uk

Abstract. AES, ElGamal are two examples of algorithms that have been developed in cryptography to protect data in a variety of domains including native and cloud systems, and mobile applications. There has been a good deal of research into the use of side channel attacks on these algorithms. This work has conducted an experiment to detect malicious loops inside Flush+Reload and Prime+Prob attack programs against AES through the exploitation of Hardware Performance Counters (HPC). This paper examines the accuracy and efficiency of three machine learning algorithms: Neural Network (NN); Decision Tree C4.5; and K Nearest Neighbours (KNN). The study also shows how Standard Performance Evaluation Corporation (SPEC) CPU2006 benchmarks impact predictions.

Keywords: Side-Channel Attack, Machine Learning, Flush+Reload, Prime+Probe, AES

1 Introduction

Data sensitivity has assumed increasing importance, and this is particularly true in cloud computing. The primary use of cryptographic techniques on the Internet and in cloud systems is to protect such sensitive data as, among others, patient records, banking transactions and social web accounts and posts. There have been consequent attacks, designed to steal sensitive data, that target such critical cryptographic elements as secret keys, look-up tables and mathematical operations including square multiplication.

The use of machine learning has been studied in a variety of domains, with particular emphasis on anti-virus work to protect individual computers, Intrusion Detection Systems (IDS) to provide greater network security, and spam detection to improve security of information. Machine learning filters out the noise, enabling complex and noisy datasets to be categorised. We therefore proposed a study that will compare three popular machine learning methods (NN; C4.5;

and KNN) to establish which will achieve the highest classification level in order to detect

Flush+Reload (FR) and Prime+Probe (PP) attacks, making use of malicious activities performed during the attack stages. The attacks target Last Level Cache (LLC) and we will use machine learning approaches that rely on hardware features indicating the state of CPU.

The focus of this paper is on the ability of machine learning methods to detect loops that can be used for side channel attacks and other malicious attacks, relying on Model Specific Registers (MSR). In a multicore system, a large number of processes share a limited number of cores, which means that every process will have access to the components inside the core in which it runs, including MSR, whether or not accessed material is connected with the process in question. It follows that an increase in workload will affect detection models, so this paper addresses the question: Does an increase in workload negatively impact detection rates?

SPEC CPU2006 is used to create a range of scenarios featuring different int and floating workloads, which generate noise, to show what impact they have, for both FR and PP attacks, on the detection model. In addition, this paper compares for accuracy and efficiency (measured by how fast they detect an attack) of the machine learning methods: C4.5, KNN and NN.

The organisation of this paper is as follows. Section 2 examines previous research into attacks and their detection., Section 3 describes the theoretical concepts involved in the three algorithms used in our research. Our findings appear in Section 4; discussion and analysis of the results are in Section 5; and Section 6 contains the conclusion.

2 Background Related Works

This section examines previously suggested techniques and methods for both attacks and the detection of attacks, for which a number of approaches have been used. This section also examines the most important of the key components involved in attacks and in defence against attacks.

2.1 Performance Monitor Unit (PMU)

Modern CPUs contain a PMU to make it possible for programmers to monitor the execution of a program or of a specific piece of code within the program [7]. Multicore processors contain one PMU in each core. A PMU comprises registers (called Performance Monitoring Counters (PMCs)) that maintain account of events of different types that occur within the CPU. There are more than 200 such events and they include: cache misses; elapsed and retired instructions;

branch predictions; and hardware stalls. In a Sandy Bridge processor, the PMCs comprise two sets of counters: three fixed-function counters to count core cycles, reference cycles and core instructions; and four general-purpose counters capable of counting any events supported by the CPU model. The processor can also select four specific performance-related events to be monitored simultaneously, and can monitor more than four counters through multiplexing. There is, however, an overhead inherent in multiplexing mode due to switching between counters.

2.2 Side Channel Attacks

The first practical attack against the cryptographic algorithm DES occurred in 2003, and was proposed by Tsunoo et al. [18]. This kind of attack normally operates by stealing secret keys and depends on vulnerabilities in the cryptographic algorithms rather than a brute force attempt to apply all possible key combinations. CPU designers attempt to conceal the fundamental details of CPU components, but researchers with malicious intent have found vulnerabilities in CPU components and exploited them for side channel attacks. These vulnerabilities can be exploited through weaknesses in such OS features as memory deduplication and shared libraries[17] [4]. Software companies, having found that attacks of this sort exploit features that exist to accelerate performance, have disabled such desirable features of their software as page sharing between two processes and memory deduplication.

CPU cycles were the original key factors in both attack and countermeasures (2004-2010). Attackers sought to monitor such processes as cache utilisation by the use of CPU cycles. Measuring their activities made it possible to use statistical methods to deduce cache lines recently used by the intended victim.

In the early days of side channel attacks, the most frequent target was L1 cache with low bandwidth in 2004 [1], 2005 [15], 2006 [14], 2009 [16], in 2012 [22]. In 2014, Yarom et. al [20] used LLC to obtain a fast attack to retrieve over %90 of key bits. Then Irazoqui et al. [9] proposed even faster attack and the whole key bits became recoverable in less than one minute in the cloud systems. Furthermore, Irazoqui et. al [8] made it possible through huge pages for attackers to obtain physical addresses. Where pages of 2MB rather than 4KB are enabled, and the data put into CPU caches, the attacker can deduce entire physical addresses.

2.3 Flush+Reload (FR)

This mechanism, first named time+evict and applied on L2 cache, was discovered in 2011 by [5]. Three years later, Yarom et al [20] documented a Flush+Reload attack against RSA applied on inclusive LLC and capable of recovering 96.7% of the bits from secret keys between isolated processes. That same year, Irazoqui et al [9] showed the ability to recover the secret key in less than one minute.

We applied a Flush+Reload attack against the OpenSSL implementation of AES from [9]. AES is a T-table based algorithm. This table [1] can be shared between unrelated processes on the same system, meaning that attacker and intended victim can share the same page. The attack requires page sharing and for static memory to be set by disabling Address Space Layout Randomisation (ASLR). LLC acts as a covert channel between the attacker and victim processes. The attacker uses the `cflush` instruction to observe the victims processes. The attacker begins by flushing one line in LLC and then scans the range of addresses in the T-table. Examining the location of the T-table in `libcrypto.so` file enables the attacker to find the tables start and end addresses. A short access time means a hit; otherwise this is a miss.

In a shared library, the attackers action in flushing cache line(s) removes all data in CPU caches, since attacker and victim are accessing the same shared file. The victim must therefore bring data back to the CPU caches. The result is local core events such as cache misses.

2.4 Prime+Probe (PP)

Cloud service providers disable memory deduplication to prevent FR attacks, but attackers use PP to carry out side channel attacks because PP requires neither page sharing nor ASLR to be enabled. Instead, the attacker evicts specific set(s) of data from cache memory and waits for a victim to evict its monitored set(s); when access time is compared, a fast access means that the cache line has been touched. The details can be found in [8]

2.5 Detection and Mitigation

A number of approaches have been used for the detection of side and covert channel attacks. Zhang et. al. [21] proposed a statistical analysis of cache access-driven attacks mainly that would rely on CPU cycles to monitor accessed and non-accessed cache-based (miss/hit) attacks. Briongos et al. [2] suggested detection based on FR attack against AES by analysing the flush instruction of multiple cache lines that forms the core of the attack. Their model relied for the most part on the CPU cycle as a primary source for data collection.

In another approach, PMU registers were used to give greater granularity so that features supporting detection mechanisms could be extracted at higher resolution. Zhang et al. [21] proposed CloudRadar, where detection of signatures and anomalies detect existing and new forms of side channel attacks as well as such other cache attacks as denial of service attacks against CPU caches.

[1] The shared library that OpenSSL produces during compilation is libcrypto.so

Kayaalp et al. [10] proposed Relaxed Inclusion Caches (RIC) to mitigate side channel attacks, while Vogl et al. [19] suggested PMC-based trapping as a way of monitoring programs at the instruction-level within VMs. The authors showed that it was possible to monitor a specific instruction in cloud systems.

More recently, Nomani et al. [13] proposed a detection and mitigation mechanism by injection (integrating or hooking) the OS system scheduler to monitor memory usage to detect the existence of malicious programs. The author primarily focused on integer and floating-point units and their effectiveness in measuring CPU such component usage as the CPU cache. They injected the OS scheduler to deploy the NN machine learning algorithm in user-space and collected data through the kernel from the PMC registers to predict malicious processes by identifying and separating two processes that are memory intensive.

3 Methodologies

In this work, we present three common supervised algorithms to classify Flush+Reload and Prime+Probe side channel attacks against AES, and then compare the results of each method used to determine which one most efficiently detects the attack.

3.1 Principle Component Analysis (PCA)

PCA is an unsupervised machine learning algorithm widely used in dimensional reduction to facilitate classification. It is a simple and widely used algorithm which finds the direction of spread of data with the greatest variance and then generates new coordinates.

3.2 Neural Network (NN)

NN is a supervised machine learning algorithm. It can build a predictive model by learning from historical data the patterns to throughput binary or multiclass classification.

The ability of NN to self-learn from examples allows researchers to train NN with features from CPU events from which it acquires the knowledge to classify CPU activities into malicious and non-malicious. Neural network architecture can generally be categorised into: single-layer feed-forward network; multi-layer feed-forward network; and recurrent network. A number of other types have emerged, however, including: perceptron; backpropagation; self-organising map; adaptive resonance theory; and radial basis function.

Ngiam et al. [12] showed the efficiency of the algorithm in dealing with low dimensional data sets. This can work more efficiently with PCA that reduces the dimension of our data. To accelerate the learning process, we use PCA to reduce the dimension and then pass it to the optimisation algorithm L-DFGS, which is efficient for small data sets.

In choosing between three activation functions, we have considered speed and accuracy. Because such attacks are fast, data can be retrieved in less than one minute. Recently, Kingme et al. [11] introduced Adaptive Moment Estimation (ADAM), an optimisation technique used in NN that is fast, computationally efficient and requires less memory than DFGS. It also deals efficiently with large data sets. The Quasi-Newton method, on the other hand, is computationally expensive and requires more memory to store the Hessian matrix, while LDFGS accelerates the speed [3] of deep network learning. They used the algorithm for large data sets and showed it to be faster than the SGD algorithm. Limited-memory DFGS does not store Hk and is therefore faster than DFGS. Faced with a large data set, as we mentioned, ADAM is faster.

3.3 K Nearest Neighbour KNN

KNN is a non-parametric classifier and a lazy learner classification method. Each tested data class is predicted by measuring the similarity of test data and training set records. Broadly, in KNN the classification process carried out for the test data relies on its neighbour, compared with K closest training example where the output is the class membership. Any sample of data points can be classified by a majority vote of its neighbour. K represent a very small integer, so if k=1, then k is assigned to the class of single nearest neighbour. Functions available for use in similarity calculations include Euclidean, Hamming, Manhattan and Minkowski.

3.4 C4.5

C4.5 is one of the oldest supervised machine learning algorithms. It is uniquely easy to read and understand. The goal is to build a model that predicts the value of a target variable by asking multiple linear questions one by one to create a boundary. Future data is classified using a very simple data structure which is called Tree.

It is a statistical classifier like other classifiers, and uses a set of data to train and build a decision tree model using the concept of information entropy. The trained data is split into n-dimensional vectors which represent features of the sample data and its class. C4.5 selects the most efficient features to divide its set of samples into subsets boosted in one class or the other using the following equation:

$$E(S) = \sum_{i=1}^{n} -P(C_i) * log_2 P(C_i) \tag{1}$$

$$G(S,F) = E(S) - \sum_{i=1}^{m} P(F_i) * E(S_{Fi}) \tag{2}$$

When, $S = \{s_1, s_2, ...s_n\}$ represents the set of samples. $E(S)$ is an informational entropy of S, $G(S,F)$ is a function to gain S after splitting feature F. $Pr(Ci)$ calculates the frequency of class Ci in S. The function $E(S_{FI})$ generates a subset of S with items that have F_i value.

4 FR and PP Attack Detection

In this work, we demonstrate the hardware specifications used for data collection.

4.1 Hardware and Software Specifications

The experiment was conducted on HP Proliant DL360 G7 with Intels Xeon X5650 2.66GHz processor with 16 GB RAM running Ubuntu 14.04. The various tests used SPEC cpu2006.

4.2 Experiment

In this section, we conduct an experiment study by using data collected from the experiment. We create an agent process that encrypts fake data with the intention of simulating a victim, and used the custom Loadable Kernel Module (LKM) to access PMC registers with minimum overhead in order to gain high resolution data. Our data set consists of 7 features, three fixed function registers (core cycles, reference cycles and core instructions) and four more efficient programmable events. For this experiment, we selected the most efficient events having a positive impact on the classification of the selected methods by considering their relationship to attacks. We collected 100 samples, which is optimal, $S = \{S_1, S_2, ...S_{100}\}$ and for each S_i we flattened all features into a single row (one big vector of mixed data). Each sample is so arranged that $S_1 = \{X_{(1,1)}.....X_{(1,n)}, X_{(2,1)}.....X_{(2,n)}, , ...X_{(7,1)}.....X_{(7,n)}, y\}$. n is the number of encryption iterations executed by the victim to collect the specified event. In this experiment, we used 3000 iterations for each event. y is the binary class which represents attack or normal.

The data is collected under two scenarios, one for light and one for heavy workloads. In the first scenario, high resolution data was ensured by running only victim and attack programs. In the second scenario, we added noise by running additional applications from SPEC SPEC2006[2], two int applications (`bzip2` and `gcc`) and two floating applications (`bwaves` and `dealII`)

The dataset was split into training and testing sets to prepare for machine learning algorithms. Training sets contain 80 samples and 20 testing sets. To determine the influence of different data set splits under each method, we split the data sets randomly into 20-fold cross validations.

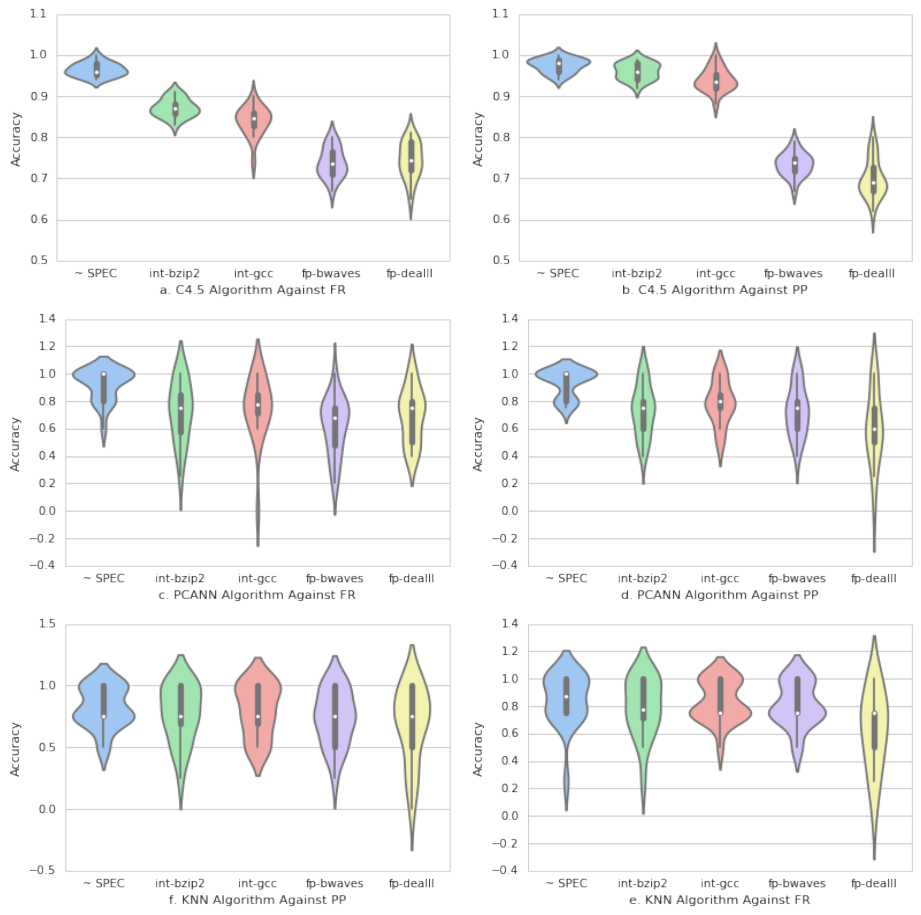
In this study, we show the impact of malicious loops running inside FR and PP attack programs on the victims processes, which use a cryptographic algorithm to encrypt sensitive data. Our hope was to detect the attack in both light and heavy workloads. The attacker would try to interfere with the victims processes and synchronise itself on the shared LLC by monitoring its cache memory activities and using statistics to deduce the cache lines most recently used by the victim. We also hoped to detect the attack in the shortest possible time less than 5 second; when the efficient attack [9] requires over 50 seconds to recover the whole key bits. This experiment can be applied in cloud systems, except for the additional overhead, which is produced by an additional translation layer. This definitely reduces the resolution rate detection, as the most recent detection work [6][2][21] shows the difference in accuracy rates between native and cloud systems.

The shared library co-allocates two unrelated processes on LLC on the same machine. Thus, we would detect malicious FR and PP attack activities when a malicious loop is run to synchronise with the victim process in order to give the attacker a chance of accessing the shared memory. The aim of our hypothesis was to evaluate the best classification method and the impact of SPEC in detecting such attacks with a high rate of accuracy even with the loading of the benchmark.

4.3 Result Analysis and Discussion

We are looking for an optimal classifier that works most accurately and efficiently among selected methods under both light and heavy workloads. Each of the three algorithms presented in the previous section was run on each of the 20-fold splits of the data set into training and testing sets. Based on previous studies and the results gained from our experiment, we compare the methods based on accuracy and efficiency because these two factors are important to victims when dealing with sensitive data. For accuracy, the victim needs to correctly classify the attack, while for efficiency, the victim needs to detect the attacks quickly before the attacker retrieve the whole key-bits and disrupt the attack.

[2] SPEC SPEC2006 is widely used to evaluate performance of computer systems https://www.spec.org/



Figuer 1. Comparison Accuracy rate of C4.5, PCANN and KNN.

Each violin shape represents the sample distribution of 20 fold across validations of the experiments. The accuracy rate of detecting FR and PP attacks are compared under different workloads without SPEC ($\sim SPEC$), int-bzip2, int-gcc, fp-bwaves and fp-dealII. The inflated area indicates the density of the accuracy rate in 20 fold cross validation. Inside each violin, the accumulated black dots in the middle represent the accuracy of the tests and the white dot represents the median of the accuracy over all the tests.

The results, presented in Table (1) and Figure (1), show the accuracy of side channel attack classification including FR and PP techniques for all methods in three scenarios without SPEC ($\sim SPEC$), CPECint or SPECfp. The C4.5 algorithm performs with highest accuracy in all scenarios in detecting FR. Without SPEC the success rate is 0.97%. This falls in SPECint to 0.91% and 0.87% in `bzip2` and `gcc` respectively. There is a further decrease to 0.74% in SPECfp to 0.74% and 0.75% for `bwaves` and `dealII` respectively. However, in PP detect-

Table 1. Classification Accuracy for the three methods C4.5, PCANN and KNN, against two attacks Flush+Reload (FR) and Prime+Probe (PP).

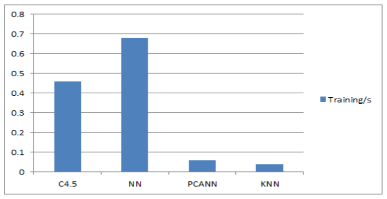
Classification Accuracy on FR and PP							
SPEC	Benchmarks	C4.5		NN		KNN	
	Attack	FR	PP	FR	PP	FR	PP
	No SPEC	0.97	0.98	0.93	0.76	0.85	0.83
SPECint	bzip2	0.91	0.96	0.8	0.8	0.8	0.78
	GCC	0.87	0.94	0.77	0.79	0.84	0.8
SPECfp	bwaves	0.74	0.74	0.73	0.73	0.73	0.74
	dealII	0.75	0.7	0.7	0.64	0.63	0.7

ing it classifies better in SPEC, `bzip2` and `gcc`. Stays the same in `bwaves`, but it is worse in `dealII`. This is because, in a PP attack, the attacker uses more CPU components and this maximises the number of occurrences of specified events.

PCANN is good at detecting FR without SPECint and SPECfp, but performs poorly in detecting a PP attack even without benchmarks. KNN has a similar accuracy rate for FR and PP attacks, but drops down in a PP attack. The results from C4.5 are therefore seen to be more reliable and robust than from PCANN and KNN. This because C4.5 method deals with noisy data better than the rest of the methods due to fast data exploration and find the relation between the most significant variables. Turning to efficiency, Decision Tree is more inefficient than PCANN and KNN, but still detects the attack with reasonable efficiency.

However, running `bzip2`, `gcc` applications in order to load the SPEC benchmark showed C4.5 to have a higher level of accuracy than either NN or KNN. KNN efficiency was slightly lower than C4.5, but NN had the worst accuracy. When `bwaves` is loaded, they all have poor accuracy, because `bwaves` is in the float application group and floating operations make heavier use of CPU components than integer operations, resulting in a high number of cache misses and degrading the training of the classification models.

The results shown in Figure 2 indicate that the size of the data set will be enough for the detection agent to be able to learn malicious activities in a very short time; the worst case is less than 1 second and compares well with recent and fast FR attack by [9] that needed over 50 seconds to retrieve the entire key bits. The agent can detect the attack early enough to prevent the attacker stealing the whole key and can perform the actions necessary to stop the attacker.



Figuer 2. Comparison of time execution for training in selected classifiers

It follows that detection of FR and PP attacks will be difficult in noisy environments, and especially so when intensive floating-point applications are running. This is because floating point applications make use of CPU caches and generate a large number of cache misses, while detection relies partly on CPU cache misses to detect FR and PP attacks.

These methods can be used by a host OS, in both native and cloud systems (though it must be noted that cloud systems are less accurate than native systems) to distribute a fake process running cryptographic algorithms such as AES to identify malicious activities and prevent them from stealing the whole of a secret key.

The results show that system activities in the background do not significantly impact the results, with all methods performing well, but intensive workloads introduce more noise into the system and have a negative impact on the accuracy. In particular, SPECfp benchmark made the result worse than SPECint benchmarks, because the floating operations cause high occurrence of CPU events. Loading SPEC benchmarks places stress on CPU components, and particularly on caches. A SPECfp benchmark interferes with the monitoring processes and introduces noise to the environment.

5 Conclusions

We investigated the use of three classification methods for detecting Flush+Reload and Prime+Probe attacks in two different scenarios, one with and one without SPEC CPU2006 benchmark workloads. We concluded that a heavy workload has a negative impact on the detection rate due to stress on the CPU components. Our findings indicated that the Decision Tree classifier is better than NN and KNN to detect Flush+Reload and Prime+Probe attacks.

References

1. Daniel J Bernstein. Cache-timing attacks on aes, 2005.
2. Samira Briongos, Pedro Malagón, José L Risco-Martín, and José M Moya. Modeling side-channel cache attacks on aes. In *Proceedings of the Summer Computer Simulation Conference*, page 37. Society for Computer Simulation International, 2016.
3. Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.
4. Daniel Gruss, David Bidner, and Stefan Mangard. Practical memory deduplication attacks in sandboxed javascript. In *European Symposium on Research in Computer Security*, pages 108–122. Springer, 2015.

5. David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games–bringing access-based cache attacks on aes to practice. In *2011 IEEE Symposium on Security and Privacy*, pages 490–505. IEEE, 2011.
6. Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar. Cache-based application detection in the cloud using machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 288–300. ACM, 2017.
7. Intel Intel. and ia-32 architectures software developers manual. *Volume 3A: System Programming Guide, Part*, 1(64), 64.
8. Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. S $ a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes. In *2015 IEEE Symposium on Security and Privacy*, pages 591–604. IEEE, 2015.
9. Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! a fast, cross-vm attack on aes. In *International Workshop on Recent Advances in Intrusion Detection*, pages 299–319. Springer, 2014.
10. Mehmet Kayaalp, Khaled N Khasawneh, Hodjat Asghari Esfeden, Jesse Elwell, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. Ric: Relaxed inclusion caches for mitigating llc side-channel attacks.
11. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
12. Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.
13. Junaid Nomani and Jakub Szefer. Predicting program phases and defending against side-channel attacks using hardware performance counters. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*, page 9. ACM, 2015.
14. Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology–CT-RSA 2006*, pages 1–20. Springer, 2006.
15. Colin Percival. Cache missing for fun and profit, 2005.
16. Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.
17. Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory deduplication as a threat to the guest os. In *Proceedings of the Fourth European Workshop on System Security*, page 1. ACM, 2011.
18. Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of des implemented on computers with cache. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 62–76. Springer, 2003.
19. Sebastian Vogl and Claudia Eckert. Using hardware performance events for instruction-level monitoring on the x86 architecture. In *Proceedings of the 2012 European Workshop on System Security EuroSec*, volume 12, 2012.
20. Yuval Yarom and Katrina Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, 2014.

21. Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 118–140. Springer, 2016.
22. Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.